Universidad Zaragoza
1542

Trabajo Fin de Grado

Sistema para categorización de textos en un ámbito nicho con pocos datos etiquetados

Miguel Escribano Pérez

Director: Jorge Gracia del Río
Codirector: Ricardo J. Rodríguez Fernández

Grado en Ingeniería Informática

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Febrero 2021
Curso 2020/2021

Resumen

El incremento exponencial de la generación de contenido en Internet ha obligado a automatizar tareas de gestión que antes eran realizadas por humanos, lo cual ha impulsado un gran desarrollo de las técnicas de Inteligencia Artificial. Estas nuevas herramientas pueden ser de ayuda a la hora de moderar contenidos peligrosos difundidos en redes sociales, como la apología de los trastornos alimenticios.

En este trabajo se colabora con la Fundación APE para implementar un clasificador de texto que detecte la promoción de la anorexia y la bulimia en mensajes de Twitter. Dicho clasificador será integrado en un software de seguimiento de interacciones en redes sociales para monitorizar en tiempo real la difusión de estos contenidos. El objetivo principal de este trabajo es estudiar las herramientas de código abierto disponibles más relevantes para dicha tarea de clasificación de textos y compararlas en el contexto concreto que nos ocupa (detección de mensajes promotores de trastornos alimenticios).

Adicionalmente, se ha generado un corpus de textos etiquetados como promotores o no promotores de trastornos alimenticios expandiendo un corpus preexistente con mensajes recolectados de Internet. Con él se han entrenado clasificadores de texto basados en cinco herramientas de procesamiento del lenguaje natural distintas, FastText, SpaCy, Transformers, Custom_BoW y Custom_TF-IDF. Estas dos últimas han sido implementadas manualmente como *baseline* de la comparativa. Además, se han aplicado distintas formas de preprocesado de texto, incluido un corrector ortográfico propio, para reducir el ruido en las muestras.

Los resultados obtenidos muestran una clara superioridad de las herramientas Transformers y FastText, que han superado el 0.95 de *F1-score*, siendo mejores que los logrados por los otros clasificadores estudiados. En concreto, FastText se considera el modelo más adecuado en este caso de estudio por su excelente equilibrio entre rápido tiempo de respuesta y calidad del clasificado. Los resultados más consistentes se han logrado con técnicas de preprocesado de textos poco intrusivas y se desaconseja el uso de correctores ortográficos por su impacto en el tiempo de respuesta, que no resulta en mejoras notables en la calidad de los resultados.

Como conclusión de este trabajo, se ha comprobado que es viable categorizar texto en lenguaje natural con un corpus reducido de ejemplos, sin hardware dedicado ni conocimiento extenso de Inteligencia Artificial. Trabajos futuros deben abordar cómo mejorar la calidad de etiquetado de los corpus de datos generados, investigar si los resultados obtenidos son similares en otros casos de clasificación de texto, y estudiar el uso de *Machine Learning as a Service*, como en el caso de OpenAI con GPT-3.

Palabras clave: anorexia, bulimia, trastornos alimenticios, redes sociales, aprendizaje automático, procesamiento de lenguaje natural, redes neuronales

Abstract

The exponential increase in the generation of content on the Internet has forced the automation of management tasks that were previously carried out by humans, thus driven a great development of Artificial Intelligence techniques. These new tools are helpful to moderate dangerous content spread on social networks, such as the apology for eating disorders.

In this work, we collaborate with the APE Foundation to develop a text classifier that detects the promotion of anorexia and bulimia in Twitter messages. This classifier will be integrated into a software system for monitoring interactions in social networks to detect the dissemination of these contents in real time. The main goal of this work is to study the most relevant open-source tools available for this task of text classification and to compare them in the specific context at hand (detection of messages promoting eating disorders).

Additionally, a corpus of texts labeled as promoters or non-promoters of eating disorders has been generated by expanding a pre-existing corpus with messages collected from the Internet. With it, text classifiers based on five different natural language processing tools have been trained. Specifically, FastText, SpaCy, Transformers, Custom_BoW and Custom_TF-IDF. These last two have been implemented as baseline of the comparison. Additionally, various forms of text pre-processing have been applied, including an original spell checker, to reduce noise in samples.

The results show a clear superiority of the Transformers and FastText tools, which have exceeded the 0.95 of F1-score, being better than those achieved by the other classifiers studied. Specifically, FastText is considered the most appropriate model in this case study due to its excellent balance between fast response time and quality of the results obtained. The most consistent results have been achieved with low-intrusive text preprocessing techniques and the use of spell checkers is discouraged due to its impact on response time, which does not result in notable improvements in the quality of the classifications.

As a conclusion to this work, it has been proven that it is feasible to categorize text in natural language with a reduced corpus of examples, without dedicated hardware or extensive knowledge of Artificial Intelligence. Future work is needed to improve the labeling quality of the generated data corpus, investigate whether the results obtained are similar in other cases of text classification, and study the use of Machine Learning as a Service, as in the case of OpenAI with GPT-3.

Keywords: anorexia, bulimia, eating disorders, social networks, machine learning, natural language processing, neural networks

Agradecimientos

A Ricardo y Jorge, por guiarme en este proyecto. Este trabajo me ha abierto los ojos a un área fascinante de la informática y no podría haberlo hecho sin su apoyo.

A la Fundación APE por sus esfuerzos por proteger a las víctimas de Ana y Mia. Espero que mi pequeña contribución sea de ayuda para evitar que más familias pasen por ese infierno.

A mi familia, por seguir ahí para mi hasta el día de hoy, a pesar de los rodeos en el camino.

A Pilar López-Úbeda y su equipo de la Universidad de Jaén por permitirme construir sobre su investigación.

Al I3A por proveer los recursos técnicos para ejecutar los experimentos.

Y a todas y cada una de las personas que contribuyen al desarrollo de proyectos de código abierto. Su trabajo incesante nos da alas para construir el futuro.

Índice general

Índice de figuras

Índice de cuadros

Índice de algoritmos

1. Introducción

La creciente popularidad de Internet desde principios de siglo ha dado lugar a un incremento exponencial de la generación de contenido, inicialmente textual y más tarde audiovisual. Esto se materializa en que en sólo diez años la cantidad de información en Internet ha crecido de 2 zettabytes en 2010 a 59 zettabytes en 2020, y se estima que en 2024 se superarán los 149 zettabytes[1]. Tareas de gestión que antes eran realizadas por humanos, como la revisión, corrección e indexación de contenidos en la Web, pasaron a ser inasumibles sin la ayuda de computadores. Esta necesidad de delegar en los ordenadores tareas complejas, unida al incremento de la capacidad de computación disponible, ha llevado a un rápido desarrollo de las técnicas de inteligencia artificial, sobre todo las de aprendizaje automático y profundo. En apenas unas décadas se pasó de usar sistemas basados en reglas y estadísticas a entrenar redes neuronales capaces de comprender las sutilezas del lenguaje humano. No sólo se han mejorado las herramientas, sino que también se han hecho más accesibles, permitiendo que individuos y organizaciones pequeñas hagan uso de tecnologías antes sólo disponibles para multinacionales e institutos de investigación.

Sin embargo, a pesar de los beneficios del acceso universal a Internet y los avances tecnológicos derivados, la proliferación de foros y redes sociales ha traído consigo la aparición de comunidades cuyos miembros comparten comportamientos nocivos o directamente peligrosos para sí mismos y los demás. Un ejemplo de este tipo son los movimientos *pro-ana* y *pro-mia*, que promueven entre los jóvenes trastornos alimenticios como la anorexia (personificada con el nombre Ana) y la bulimia (denominada Mia) (Lladó, Gonzalez-Soltero, y Valderrama 2017). Estas comunidades no sólo transmiten una visión distorsionada y enfermiza del ideal del cuerpo propio, sino que comparten técnicas de adelgazamiento peligrosas para la salud e incitan a los adolescentes al distanciamiento social de familiares y amigos. A pesar de que actualmente no existe una legislación que persiga este tipo de movimientos, es muy importante hacer un seguimiento exhaustivo de su evolución para llevar a cabo tareas de prevención.

El presente trabajo se ha realizado en colaboración con la Fundación APE[2], una organización sin ánimo de lucro que apoya a pacientes de trastornos alimenticios y a sus familias, la cual está desarrollando un sistema informático para analizar en tiempo real las redes sociales y determinar las relaciones de influencia entre usuarios que difunden contenidos *pro-ana* y *pro-mia*. Dicha colaboración consiste en desarrollar un clasificador de texto que se integre con el software desarrollado por Revillo Rey (2020), el cual recoge mensajes de redes sociales, los clasifica como promotores o no promotores de trastornos alimenticios y genera grafos de interacción entre los usuarios involucrados. En concreto, en el presente trabajo se estudian las diversas herramientas de análisis de texto disponibles de forma pública y su utilidad a la hora de categorizar mensajes de redes sociales relacionados con la apología de trastornos alimenticios.

1.1. Objetivos y contribución

La Fundación APE, al igual que muchas otras organizaciones pequeñas, no dispone de grandes recursos ni experiencia interna en inteligencia artificial, por lo que es inviable desarrollar soluciones avanzadas a medida. Por otro lado, no dispone de un conjunto de textos de ejemplo de los mensajes a categorizar y

[1] Datos extraídos de la gráfica "Total data volume worldwide 2010-2024" de Statista (2020)
[2] https://fundacionape.org/

se desconoce a priori si existe un corpus de ese tipo en castellano, por lo que es de esperar que el que se use para entrenar los sistemas, ya sea generado internamente o por un tercero, sea de un tamaño reducido. Por último, el clasificador de textos va a ser parte de un sistema de análisis en tiempo real, por lo que la eficiencia temporal es un factor a tener en cuenta.

Por todo esto, el principal objetivo del presente trabajo es identificar herramientas de clasificación de texto de código abierto que la Fundación pueda usar de forma gratuita para detectar textos promotores de anorexia y bulimia, y comparar sus resultados, tanto en calidad de la clasificación como en eficiencia temporal, al ser entrenados con un corpus de datos reducido. La conclusión de dicha comparativa será una propuesta de clasificador a integrar en el sistema de la Fundación APE.

Una tarea adicional necesaria para la consecución de dicho objetivo es la búsqueda o recolección de un corpus de textos etiquetados como promotores y no promotores de trastornos alimenticios con el que entrenar los clasificadores.

1.2. Estructura del documento

En los capítulos 2 y 3 se presentan, respectivamente, un resumen de los conceptos teóricos que sirven de base para este trabajo y un estado de la cuestión sobre la detección de trastornos psicológicos en textos. A continuación, en los capítulos 4 y 5, se presentan la metodología de experimentación y la forma en que se ha generado el corpus de datos de entrenamiento, respectivamente. Después, el capítulo 6 detalla las distintas opciones y componentes de los clasificadores evaluados, cuyos resultados experimentales se discuten en el capítulo 7. Finalmente, el capítulo 8 recoge las conclusiones del presente trabajo así como ideas de trabajo futuro.

2. Conceptos previos

En este capítulo se van a presentar de forma resumida los conceptos técnicos generales necesarios para entender el resto del trabajo. En primer lugar, se introduce el campo del procesamiento del lenguaje natural. Luego, se describen algunos métodos de representación de información textual. Después se discute la aplicación de redes neuronales en el procesamiento de texto. Por último, se aborda la explicabilidad de modelos de aprendizaje profundo.

2.1. Procesamiento de lenguaje natural

El procesamiento de lenguaje natural (NLP por sus siglas en inglés) es un campo de la inteligencia artificial relacionado con la comprensión y generación de lenguaje humano por parte de un computador (Jurafsky y Martin 2009). Algunas tareas de NLP son la categorización de un texto, el calculo del árbol sintáctico de una frase, la identificación de a quién se refieren los pronombres en cada momento a lo largo de un párrafo y la identificación de las emociones que transmite un mensaje.

Hay multitud de técnicas de inteligencia artificial que pueden ser usadas en NLP como reglas expertas, métodos estadísticos, regresiones lineales y no lineales, y redes neuronales. Al igual que en otras áreas de la inteligencia artificial, el entrenamiento de sistemas de NLP puede ser supervisado (entrenado con un corpus de datos etiquetados previamente), no supervisado (entrenado con un corpus de datos sin etiquetar) y semi-supervisado (entrenado con un corpus con unos pocos datos etiquetados).

2.2. Representación de la información textual

Los textos que entran a un sistema basado en NLP pueden ser heterogéneos y de longitud variable, tanto en número de palabras como de caracteres por palabra. Por otra parte, por la forma en que se implementan muchos sistemas de inteligencia artificial, estos requieren estructuras de entrada estables (el sistema sabe por adelantado qué esperar), por lo que es necesario aplicar técnicas de normalización. A continuación se enumeran algunas de las técnicas usadas para normalizar el texto que va a ser suministrado al sistema con el fin de homogeneizar los datos, reducir ruido, etc.

El primer proceso aplicado normalmente es la **tokenización**, consistente en dividir el texto en unidades más pequeñas llamadas *tokens* mediante delimitadores (saltos de línea, espacios, signos de puntuación, etcétera). Los tokens resultantes pueden ser párrafos, frases, palabras o sílabas, dependiendo del propósito concreto y los procesados posteriores. Estos tokens pueden ser indexados de tal forma que cada uno esté asociado a un identificador único de tamaño constante.

Una vez se ha tokenizado el texto, este se puede representar como una secuencia de **n-gramas**. Un n-grama es una secuencia de n tokens contiguos en un texto. Al codificar un texto en n-gramas el primer n-grama corresponde a los n primeros tokens, el segundo a los n tokens a partir del primer elemento, y así sucesivamente. El uso de n-gramas de tamaño mayor a 1 (caso trivial de palabras individuales) puede ayudar a reflejar las relaciones semánticas entre los tokens del texto, mejorando los resultados posteriores. Sin embargo, la cantidad de n-gramas posibles para un conjunto de tokens crece

exponencialmente conforme se incrementa n y muchos de ellos sólo aparecerán una vez, lo que puede dar lugar a un sobreajuste en los modelos entrenados con un valor de n grande.

Una vez se ha transformado el texto en una secuencia de n-gramas, esta puede ser usada con modelos NLP que acepten entradas secuenciales. Sin embargo, en muchas situaciones es interesante aplicar técnicas de reducción de la dimensión para compactar la información en un valor de tamaño fijo que pueda ser introducido en modelos NLP no secuenciales, a costa de perder parte de la información del texto. Algunas de la técnicas de reducción de dimensiones son las bolsas de palabras y los *word embeddings*, explicados a continuación.

Una **bolsa de palabras** asocia cada uno de los n-gramas del texto con el número o porcentaje de veces que aparece. Esta estructura no conserva información sobre el orden de los elementos contenidos ni el posicionamiento relativo entre ellos, pero permite comparar la similitud de textos en función del porcentaje de aparición de los elementos. Si se dispone de textos etiquetados con categorías, es posible combinar las bolsas de palabras de los textos de una misma categoría para obtener una representación textual media de dicha categoría.

Un ***word embedding*** o vector de palabras transforma secuencias de n-gramas en vectores de números reales de dimensión fija, que suele estar en el rango entre 100 y 300. Cada una de las componentes del vector representa el nivel de expresión en el n-grama de un concepto capturado mediante aprendizaje profundo del texto. Hay distintas formas de realizar esta transformación, pero la más famosa es `word2vec`, una técnica desarrollada por Google (Mikolov et al. 2013) que usa redes neuronales para detectar las semejanzas semánticas entre palabras. Sin embargo, no es trivial identificar qué conceptos en concreto han sido capturados por el modelo o por qué y hay una gran discusión al respecto. Una expresión muy clara de la complejidad de este problema es el artículo que publicaron Goldberg y Levy (2014) analizando el algoritmo de *word2vec*, cuya conclusión comenzaba con la frase "¿Por qué produce esto buenas representaciones de las palabras? Buena pregunta. Realmente no lo sabemos".

Los *word embeddings* no sólo sirven para introducir texto en otros modelos NLP, sino que pueden ser usados como clasificadores por sí mismos. Esto se debe a que es posible comparar los vectores mediante el coseno derivado del producto escalar, de tal forma que se pueden agrupar términos con significado relacionado, que serán aquellos cuya distancia angular sea menor dentro del espacio vectorial. Por ejemplo, la herramienta FastText, desarrollada por el equipo de inteligencia artificial de Facebook (Joulin et al. 2016), genera vectores para cada una de las clases posibles y los compara con el del texto a clasificar.

2.3. Redes neuronales aplicadas a NLP

Una **red neuronal** es una estructura formada por capas de neuronas artificiales conectadas entre sí. Una neurona recibe varios datos numéricos como entrada en función de los cuales emite una salida que propaga a otras. El proceso de entrenamiento de una red neuronal consiste en introducir datos a la primera capa, recuperar las salidas generadas por la última, comparar los resultados obtenidos con los esperados, modificar los parámetros de las neuronas (el peso que se asigna a cada una de sus entradas de cara a emitir una salida) y observar si el resultado mejora. El mecanismo mediante el que se realizan estas modificaciones se llama retropropagación (Rumelhart, Hinton, y Williams 1986).

Hay una gran variedad de arquitecturas de redes neuronales, dependiendo de cómo interactúan las neuronas entre sí y con la información. A continuación se mencionan algunas de las más relevantes de cara al NLP.

Las **redes neuronales convolucionales** (CNN) son un tipo de redes sencillas en las que las neuronas de una capa sólo se conectan con algunas de las de la siguiente capa, por lo que la salida depende exclusivamente de la entrada. Versiones primitivas de CNN han aparecido en la literatura desde

principios de los años 80, pero fueron LeCun et al. (1989) los que implementaron la primera CNN con capacidad de aprendizaje mediante retropropagación como las de hoy día. En NLP se suelen usar en combinación con formas de representación textual que compacten la información de frases o párrafos, como *word embeddings*.

En las **redes neuronales recurrentes** (RNN) las neuronas pueden enviar su salida a otras de capas anteriores o a sí mismas, formando grafos direccionales. De este modo, las RNNs pueden recibir secuencias de parámetros y el procesado de un elemento tiene en cuenta el resultado del procesado de los elementos anteriores, permitiendo detectar correlaciones. Versiones tempranas de RNN incluyen las redes de Hopfield (1982), pero fue con la introducción de las redes *Long-Short Term Memory* (Hochreiter y Schmidhuber 1997) cuando se vio su gran impacto en tareas de procesamiento de lenguaje.

La forma en que las RNNs y similares trabajan puede llegar a ser muy costosa, ya que el desenrollado de los grafos que las conforman puede dar lugar a secuencias muy largas. Por ello se introduce el concepto de **red neuronal con atención** (Chorowski et al. 2014, 2015), en el cual el sistema es capaz de fijarse en los segmentos del texto con más contenido semántico e ignorar el resto.

A raíz de las redes neuronales con atención, Vaswani et al. (2017) plantearon **Transformer**, un modelo que funciona exclusivamente mediante mecanismos de atención, sin ningún otro tipo de red posterior. La arquitectura del Transformer permite procesar paralelamente los elementos del texto en un orden distinto al de entrada, lo que lo hace extremadamente eficiente y escalable, además de preciso. Los sistemas basados en Transformers como el BERT de Google (Devlin et al. 2019) y los GPT-2 (Radford et al. 2019) y GPT-3 (Brown et al. 2020) de OpenAI han pasado a ser los modelos NLP más avanzados de la actualidad. En concreto, los experimentos presentados en el trabajo de Brown et al. (2020) indican que los textos generados por GPT-3 son virtualmente indistinguibles de los escritos por humanos.

2.4. Explicabilidad de modelos

Los modelos de NLP basados en aprendizaje profundo llegan a tener miles de millones de parámetros internos, lo que hace imposible analizar de forma directa los motivos por los que un sistema entrenado toma sus decisiones. Esto es problemático tanto si funcionan bien (interesaría conocer cuál es la información relevante que ha logrado aprender) como si fallan (interesaría conocer cuál es el factor que da lugar al error).

Si bien no suele ser posible explicar el comportamiento del modelo en general, se han desarrollado técnicas para obtener explicaciones en situaciones concretas (explicabilidad local). Estas técnicas toman un texto de entrada, realizan diversas modificaciones sobre él y analizan cuánto influyen las modificaciones en los resultados de salida.

Un ejemplo de un explicador local es la herramienta LIME[1] desarrollada por Ribeiro, Singh, y Guestrin (2016), que aplica una regresión lineal para determinar cuánto peso tiene cada palabra de un texto en la categorización final. Por otro lado, la herramienta SHAP[2] del equipo de Microsoft Research (Lundberg y Lee 2017) aplica mecanismos de teoría de juegos para aproximar la influencia de cada término.

[1] https://github.com/marcotcr/lime
[2] https://github.com/slundberg/shap

3. Trabajos relacionados

Numerosos estudios han aplicado técnicas de NLP a la detección de trastornos psiquiátricos y desórdenes alimenticios. A continuación se mencionan algunas de las publicaciones más recientes al respecto.

Desde un punto de vista especializado médico, Dreisbach et al. (2019) llevan a cabo un estado del arte de las técnicas desarrolladas para detectar síntomas de distintas enfermedades en publicaciones de pacientes en redes sociales. Aunque sigue un enfoque médico mucho más amplio que cubre un amplio espectro de temas, algunos de los trabajos presentados se orientan a la detección de depresión y otras afecciones psiquiátricas, que pueden ser aplicados también a los trastornos alimenticios.

Desde un punto de vista más técnico, Spinczyk et al. (2020) aplican análisis de sentimiento mediante RNN y evalúan la intensidad de cinco emociones básicas mediante palabras clave, para después combinar la información y determinar si el paciente sufre de anorexia. Por otro lado, Ramírez-Cifuentes et al. (2020) buscan generar mejores *word embeddings* para la tarea específica de detección de pacientes de anorexia modificando `word2vec`. Estos *words embeddings* contienen una mayor información semántica para el caso concreto y dan lugar a mejores clasificadores que con las herramientas genéricas.

Una importante fuente de estudios relacionados con este tema es el eRisk[1], una competición anual destinada a la detección de trastornos psiquiátricos en redes sociales. La organización define unos objetivos y un corpus de datos que los participantes pueden usar para entrenar sus sistemas (Losada, Crestani, y Parapar 2018). La anorexia ha sido un tema recurrente, para lo que se creó un corpus de textos en inglés de Reddit, pero sólo es accesible para los equipos registrados.

Algunos de los trabajos derivados de eRisk son el de Mohammadi, Amini, y Kosseim (2019), que utiliza redes neuronales con atención para extraer características del texto y pasarlas después a una *Support Vector Machine* (SVM); Aragon, López-Monroy, y Montes (2019), que presentan el concepto de Bolsa de Sub-Emociones (BoSE) para extraer información del texto y alimentar el clasificador, obteniendo buenos resultados; Ortega-Mendoza, Farías, y Montes-y-Gómez (2019), que apuestan por el análisis de información personal en el texto (opiniones, creencias, miedos, etc.) para mejorar la detección de la anorexia; u otros trabajos que implementan clasificadores basados en redes neuronales con diferentes niveles de precisión (Ranganathan 2019; Trotzek, Koitka, y Friedrich 2018).

Los trabajos previamente mencionados parten de datasets de un tamaño considerable. Además, estos datos han sido manualmente etiquetados, ya sea por parte de la red social de la que provienen o invirtiendo un gran número de horas por parte de colaboradores. Estas situaciones no se corresponden con el presente trabajo, donde no se dispone de esta clase de recursos.

Por el contrario, López-Úbeda et al. (2019) generan desde cero un corpus de *tweets* en español etiquetados como promotores de la anorexia o no, dependiendo de si contienen ciertos *hashtags*. Con ello realizan una comparativa de la calidad de distintas técnicas de aprendizaje automático (en concreto, SVM, Bayes ingenuo, bosques aleatorios, perceptrones multicapa, regresión logística y árboles de decisión). Este último estudio comparte muchas similitudes con el presente trabajo, por lo que se ha usado como base tanto para generar el dataset como para comparar los resultados.

Un estudio interesante de cara a extender el presente trabajo es el de Yan et al. (2019), que plantean un sistema de clasificación semisupervisado en el que se recogen miles de muestras de texto y se etiquetan manualmente sólo unas decenas por parte de psicólogos, con buenos resultados. Otro área a explorar

[1] https://early.irlab.org/

de cara a futuro es la que plantean Amini y Kosseim (2020), que se centran en generar explicaciones de modelos de aprendizaje profundo de detección de anorexia con el objetivo de identificar aquellos elementos concretos dentro del texto que dan lugar al positivo, para que el profesional encargado de tratar al paciente tenga información más precisa.

4. Metodología

En este capítulo se resume el proceso seguido para la elaboración del presente trabajo. A lo largo del proyecto se ha escrito una serie de programas en Python que se encuentran en un repositorio abierto[1] para su consulta. En el Anexo A se recoge la estructura de dicho repositorio.

El primer paso ha sido obtener un corpus de textos etiquetados para poder entrenar y validar los clasificadores. Esto ha involucrado una búsqueda de corpus ya existentes y la generación de uno nuevo. Además se ha llevado a cabo un muestreo para estimar la calidad del corpus resultante. Todo esto se aborda en detalle en el capítulo 5 y los resultados se encuentran en la carpeta `dataset` del repositorio.

Algoritmo 4.1: Entrenamiento y evaluado de clasificadores

Input : corpus de textos C, lista de cadenas de preprocesadores P, lista de modelos y sus correspondientes listas de posibles parámetros M

Output: lista de combinaciones de cadenas de preprocesadores y modelos con sus métricas de validación R

```
1  foreach cadena in P do
2      CP = aplicar cadena a C
3      CPE, CPV = dividir CP en 0.8 para entrenamiento y 0.2 para validación
4      foreach modelo, params in M do
5          mejorParams = ∅
6          mejorResultado = 0
7          foreach param in params do
8              resultadoMedio = 0
9              CK = dividir CPE en 5 partes iguales
10             foreach CK' in CK do
11                 CKE = CK - CK'
12                 EK = entrenar modelo con param y CKE
13                 resultadoParcial = validar EK con CK'
14                 resultadoMedio = resultadoMedio + resultadoParcial
15             end
16             resultadoMedio = resultadoMedio / 5
17             if resultadoMedio ≥ mejorResultado then
18                 mejorResultado = resultadoMedio
19                 mejorParams = param
20             end
21         end
22         E = entrenar modelo con mejorParams y CPE
23         resultados = evaluar E con CPV
24         Añadir modelo, params y resultados a R
25         Generar evaluaciones LIME con E y mejorParams
26     end
27 end
```

[1] https://gitlab.com/miguescri/ed-classifier

A continuación, se han implementado los elementos de los clasificadores de texto a comparar, lo cual se detalla en el capítulo 6. Los clasificadores se componen de preprocesadores de texto y modelos NLP, ambos implementados y disponibles en el repositorio.

Después, se han entrenado distintos clasificadores con el 80 % de los datos del corpus combinando cada modelo con distintos preprocesadores, para después obtener métricas de validación de cada uno usando el 20 % restante del corpus. Los resultados de estos experimentos se abordan en el capítulo 7 y el código con el que han sido realizados se encuentra en el fichero `ed_classifier/train.py` del repositorio. El algoritmo de entrenamiento de los clasificadores se resume en el Algoritmo 4.1. Puesto que algunos de los modelos NLP usados disponen de parámetros, como el tamaño de los n-gramas, estos se han ajustado mediante *k-fold* de cinco pliegues optimizando el *F1-score* (esta métrica se definirá en el capítulo 6); *k-fold* es un mecanismo habitual de validación para evitar sobreajuste de parámetros y se ve reflejado entre las líneas 7 a 21.

Por último, además de obtener las métricas de validación, se han tomado diez textos del conjunto de validación y se han generado explicaciones locales con LIME[2] con cada uno de los clasificadores para visualizar las diferencias entre ellos.

Para facilitar revisiones posteriores de los datos presentados, los resultados de los experimentos se han almacenado en una base de datos SQLite y están disponibles junto a los binarios de los clasificadores entrenados y las explicaciones LIME en el repositorio.

[2]https://github.com/marcotcr/lime

5. Corpus de datos usados

Como se ha mencionado anteriormente, se ha tomado el estudio de López-Úbeda et al. (2019) como base sobre la que trabajar. En él se define la creación del corpus *Spanish Anorexia Dataset* (SAD)[1], un conjunto de *tweets* en español etiquetados como promotores o no de anorexia. Los autores han tenido la deferencia de permitir su uso en el presente trabajo.

SAD fue generado recopilando *tweets* desde la API de Twitter dada una serie de *hashtags*. Algunos de estos *hashtags* marcaban el *tweet* como promotor de la anorexia (*#anaymia*) y otros como no promotor (*#realfood*, *#comidareal* y *#fitness*). Posteriormente, los *hashtags* usados eran borrados del texto para evitar que esos términos comunes a cada clase generasen sobreajuste en los clasificadores resultantes.

A pesar de ser un importante punto de partida, en una inspección inicial del SAD se identificaron errores de etiquetado. Para estimar el porcentaje de *tweets* correctamente clasificados se llevó a cabo un etiquetado manual sobre una muestra aleatoria de 200 de estos textos y las nuevas etiquetas se compararon con las de SAD. Los resultados se recogen en el Cuadro 5.1 con los valores de verdaderos positivos (TP), falsos positivos (FP), falsos negativos (FN) y verdaderos negativos (TN), así como el porcentaje de muestras correctamente etiquetadas para SAD y para otros corpus que se detallan a continuación.

Cuadro 5.1.: Estimación de errores de etiquetado en los corpus

Corpus	**Muestras**	**TP**	**FP**	**FN**	**TN**	**Correctas**
SAD	200	64	22	3	111	87.5 %
Blogs	200	83	6	0	111	97.0 %
Blogs-tweet	200	61	23	0	116	88.5 %

Los resultados muestran que existe un problema relevante de falsos positivos en SAD. Esto se debe a que en muchos casos el *hashtag #anaymia* era el único elemento de un *tweet* que lo identificaba claramente como promotor de la anorexia y la bulimia, por lo que al borrarlo el texto pasa a ser no promotor. Por ejemplo, el *tweet* "Larga vida mis mejores amigas" está etiquetado como promotor de la anorexia, lo cual únicamente se entiende si se tiene en cuenta que originalmente el texto completo era "Larga vida mis mejores amigas #anaymia".

Con el objetivo de reducir el ruido del SAD, se ha intentado crear un nuevo corpus con menor porcentaje de error. Para ello se ha realizado una búsqueda de sitios web en español que promuevan los trastornos alimenticios, de los que se han seleccionado diecisiete blogs que tuviesen más de cinco publicaciones. Para compensar el corpus también se han añadido otros diez sitios dedicados a nutrición, deporte y trastornos alimenticios desde un punto de vista médico, de tal modo que la cantidad de textos promotores y no promotores queda equilibrada. La lista con las URIs de los blogs usados se encuentra en el Anexo B.

De cada uno de estos blogs se han recuperado todas las entradas del *feed* RSS, lo que ha dado lugar al corpus Blogs. Los 299 textos resultantes tienen una longitud muy heterogénea, por lo que son difíciles

[1] https://github.com/plubeda/SAD

de equiparar con los del SAD. Por ello, se han divididos en fragmentos de alrededor de 280 caracteres como si fueran cadenas de *tweets*, obteniendo los 1089 textos que conforman el corpus Blogs-tweet.

Al igual que se hizo con SAD, se ha realizado una estimación manual de los errores de etiquetado en Blogs y Blogs-tweet, la cual también se recoge en el Cuadro 5.1. Se observa que, si bien el corpus Blogs tiene una ratio de error muy baja por el etiquetado cuidadoso, al partir los textos en Blogs-tweet aparecen numerosos falsos positivos. Esto se debe principalmente a que muchas de las entradas de los blogs *pro-ana* (que hacen apología de la anorexia) contienen recetas de cocina que parecen normales vistas por separado sin el contexto completo. Por ejemplo, el siguiente texto es una entrada de uno de los blogs promotores en el que se describe una dieta Alisa (una de las variantes de desorden alimenticio):

> 1 semana de Alisa / Lunes / Desayuno(8:00-8:30): / Papilla de manzana, naranja y perra. / Ingredientes / 1 manzana1 pera1 naranjaAgua / Instrucciones / Lavar y pelar la manzana y la pera, partirlas en trozos pequeños. Pelar la naranja y reservarlaPoner la manzana y la pera en una olla, añadir un poco de agua y poner a fuego medio. La manzana se irá ablandando y formará una pasta. Si sobra agua, quitar un pocoAñadir el zumo de naranja y licuarlo todo. / [...]

Si se leen unas pocas frases por separado sin la línea introductoria, puede parecer una simple receta de cocina, a pesar de que al leer la dieta en detalle se ve que es claramente insuficiente desde un punto de vista nutritivo.

En definitiva, no se ha logrado el objetivo de reducir la ratio de falsos positivos en el corpus, lo cual ha de tenerse en cuenta al analizar los resultados finales. Esfuerzos futuros deberán orientarse en esta dirección para mejorar la calidad de los clasificadores entrenados. Una opción es estudiar la aplicación del trabajo de Yan et al. (2019) para etiquetar de forma semisupervisada a partir de etiquetas manuales de expertos. Otra posibilidad es hacer uso de la herramienta Cleanlab[2], que aplica el concepto de *confidence learning* para detectar etiquetas erróneas en un corpus a partir de clasificadores entrenados sobre el propio corpus (Northcutt, Jiang, y Chuang 2020).

A pesar de no mejorarse la ratio de error, añadir Blogs-tweet a SAD supone un incremento del 19 % de textos manteniendo las proporciones de etiquetas, lo cual es positivo pues se dispone de más datos de entrenamiento. El corpus resultante de la suma de SAD y Blogs-tweet se ha denominado corpus Mix y es el que se ha usado a la hora de entrenar y validar los clasificadores.

En el Cuadro 5.2 se puede ver la cantidad de textos de cada tipo que constituyen cada corpus, así como la proporción de textos promotores y no promotores. Se comprueba que las proporciones de Mix son similares a las de SAD.

Cuadro 5.2.: Distribución de muestras promotoras y no promotoras en los corpus analizados

Corpus	**Elementos**	**Promueven**	**No promueven**
SAD	5707	2707 (47.4 %)	3000 (52.6 %)
Blogs	299	134 (44.8 %)	165 (55.2 %)
Blogs-tweet	1089	540 (49.6 %)	549 (50.4 %)
Mix	6796	3247 (47.8 %)	3549 (52.2 %)

[2] https://github.com/cgnorthcutt/cleanlab/

6. Clasificadores implementados

Como se ha indicado previamente, los clasificadores se componen de un elemento de preprocesado de texto, que aplica distintos filtros a la entrada, y de un modelo NLP, que realiza la clasificación como tal. En las siguientes secciones se abordan ambas partes.

6.1. Preprocesado de texto

Por las características del texto recogido es imprescindible realizar un preprocesado que reduzca el ruido de las muestras. Para ello se han usado procesadores simples de cadenas de texto y un corrector ortográfico propio, explicados a continuación.

6.1.1. Procesadores simples

Estos procesadores eliminan o sustituyen caracteres de acuerdo a unas normas sencillas. Los procesadores implementados son los siguientes:

- `no_newlines`: elimina saltos de línea.
- `no_url`: elimina URLs.
- `separate_punctuation`: introduce espacios en blanco alrededor de los signos de puntuación.
- `same_case`: todas las letras pasan a minúscula.
- `no_accents`: trata de convertir caracteres extendidos de UTF-8 a su correspondiente carácter ASCII.
- `only_letters`: elimina todo carácter que no pertenezca al alfabeto ASCII.
- `no_anorexia_bulimia`: elimina las palabras `anorexia` y `bulimia`.
- `no_ana_mia`: elimina las palabras `ana` y `mia`, usadas habitualmente para referirse a la anorexia y la bulimia.

6.1.2. Cadenas de preprocesadores

La aplicación individual de los preprocesadores simples da lugar a resultados contraproducentes, como en el caso de `only_letters`, que fragmenta las palabras con tilde. Por ello es importante aplicarlos encadenados en el orden adecuado.

Las cadenas de preprocesadores que se han usado al entrenar los modelos son:

- `no_newlines`, `no_url`: cadena básica. Algunos de los modelos empleados requieren obligatoriamente limpiar los saltos de línea, por lo que se aplica a todos como punto de partida. Por otro lado, es muy difícil que las URL añadan valor en la clasificación, y además los procesadores posteriores las rompen, por lo que se eliminan de base.
- `no_newlines`, `no_url`, `same_case`: esta cadena busca identificar si los modelos son capaces de funcionar correctamente con texto capitalizado, o si por el contrario es necesario que todo esté en minúsculas o mayúsculas. Resultados iniciales han dado resultados favorables con esta cadena, por lo que `same_case` se ha incluido en las siguientes para obtener los mejores resultados posibles.

- `no_newlines`, `no_url`, `same_case`, `separate_punctuation`: esta cadena busca averiguar si los modelos son capaces de interpretar correctamente las palabras adyacentes a signos de puntuación.
- `no_newlines`, `no_url`, `same_case`, `no_accents`, `only_letters`: cadena orientada a detectar si los signos de puntuación y las tildes son ruido para los modelos. El procesador `only_letters` hace redundante el uso de `separate_punctuation`.
- `no_newlines`, `no_url`, `same_case`, `separate_punctuation`, `no_anorexia_bulimia`, `no_ana_mia`: el propósito de esta cadena es comprobar si ciertas palabras clave en el contexto de los trastornos alimenticios producen sobreajuste en los modelos.

6.1.3. Corrector ortográfico

Todos los preprocesadores mencionados buscan reducir el ruido de texto escrito correctamente, para así minimizar el número de *tokens* distintos presentes en la entrada al modelo. Sin embargo, los textos a analizar procedentes de redes sociales tienden a contener faltas ortográficas o errores de escritura. Esto lleva a tener muchos *tokens* en el corpus de entrenamiento que sólo aparecen una vez, lo cual puede afectar a la precisión.

Algoritmo 6.1: Corrección de texto

Input : diccionario de términos válidos D, término a corregir T, distancia de edición máxima para una corrección M
Output: término corregido TC

```
if T ∈ D then
    TC = T
else
    distancias = ∅
    foreach tVal in D do
        distancia = distancia de Damerau-Levenshtein entre T y tVal
        distancias = distancias ⋃ distancia
    end
    TCandidato, distanciaCandidata = término con menor distancia en distancias
    if distanciaCandidata ≤ M then
        TC = TCandidato
    else
        TC = T
    end
end
```

Para afrontar el problema de las palabras mal escritas inicialmente se planteó hacer uso de un corrector ortográfico de español. Sin embargo, el lenguaje usado en las webs analizadas habitualmente contiene jerga propia y anglicismos, por lo que un corrector genérico podría llegar a eliminar información relevante. Por ello se ha implementado un corrector ortográfico adaptado al problema que funciona en dos fases.

La primera fase del corrector consiste en calcular el diccionario de términos válidos. Para ello se toma el conjunto de los textos del corpus, que pueden estar previamente preprocesados para reducir el ruido, y se calcula el número de apariciones de cada término. Los términos se identifican dividiendo el texto por los espacio en blanco, por lo que los términos pueden ser palabras, números, signos de puntuación, etc. El diccionario resultante contiene todos los términos que han aparecido un mínimo de 3 veces.

Una vez calculado este diccionario, se le pasa al corrector como parámetro. A partir de ese momento, cada vez que el corrector recibe una palabra busca el término válido más parecido. Para medir cómo de

parecidos son dos términos se hace uso de la distancia de edición de Damerau-Levenshtein, que mide el número de transposiciones, adiciones y eliminaciones de caracteres necesarias para pasar de la palabra original a la de destino (Damerau 1964). El proceso se refleja en el Algoritmo 6.1. Para tratar de evitar que el corrector convierta términos desconocidos en palabras completamente distintas, también se pasa como parámetro la distancia de edición máxima a la que se puede corregir una palabra; si no hay ningún término válido que se encuentre dentro de la distancia máxima, la palabra se queda igual.

Con el objetivo de minimizar el impacto temporal de este proceso se ha implementado una memoria caché interna en el corrector que memoriza términos corregidos anteriormente.

6.2. Modelos

Los experimentos se han realizado sobre cinco modelos distintos de clasificadores NLP. Tres de ellos son herramientas de aplicación industrial, FastText, SpaCy y Transformers, mientras que los otros dos son clasificadores rudimentarios implementados en este trabajo, uno basado en bolsas de palabras y otro en TF-IDF. El objetivo de los clasificadores caseros es obtener una estimación de lo que pueda conseguir con soluciones senciallas de uso extendido y fácilmente implementables. Dichos clasificadores servirán de *baseline* para compararlos con las soluciones más sofisticadas ofrecidas por las librerías de uso profesional. A continuación se explican todos los modelos usados.

6.2.1. Bolsa de palabras

En este caso se ha implementado la bolsa de palabras `BoW` de un conjunto de `textos` como la media de las frecuencias de cada palabra a lo largo de los textos. Formalmente:

$$BoW(textos) = \Big((p_1, \frac{\sum_{t \in textos} \frac{apariciones(p_1,t)}{dim(t)}}{dim(textos)}), \ldots, (p_n, \frac{\sum_{t \in textos} \frac{apariciones(p_n,t)}{dim(t)}}{dim(textos)})\Big), \forall p \in diccionario$$

donde *diccionario* es el conjunto de los términos posibles, $apariciones(p,t)$ es el número de veces que aparece la palabra p en el texto t y $dim(t)$ es el número de palabras en el texto t.

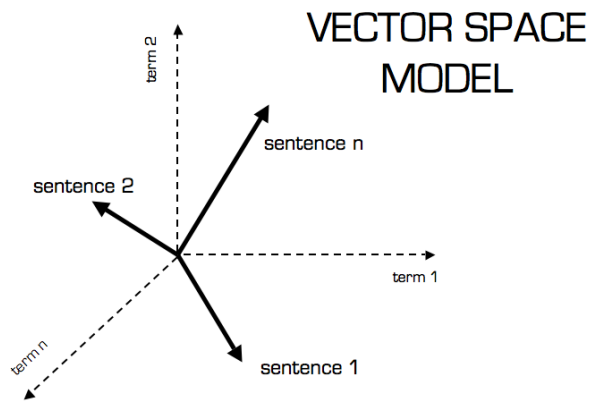

Figura 6.1.: Visualización de bolsas de palabras como vectores donde cada dimensión representa las apariciones de un término. Fuente: https://blog.christianperone.com/

Al entrenar el clasificador con textos etiquetados como promotores y no promotores de trastornos alimenticios se calculan dos bolsas de palabras, una con los textos de cada clase.

Dados un modelo entrenado y un texto a clasificar, se calcula la bolsa de palabras del texto, se compara con las bolsas preentrenadas para cada clase y se clasifica como perteneciente a la clase a cuya bolsa se parezca más. Para medir el grado en que se parecen dos bolsas de palabras se toman como si fueran vectores y se mide el ángulo entre ambos. En concreto, se hace uso de la similitud coseno derivada del producto escalar, que devuelve un valor entre 0 (distribución de palabras completamente diferente) y 1 (distribución de palabras completamente igual), definida como:

$$similitud = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

donde $A \cdot B = \sum_{palabra \in A \cap B} porcentaje(A, palabra) * porcentaje(B, palabra)$ es el producto escalar entre bolsas de palabras, definido como la suma de los productos de los porcentajes de las palabras comunes y $\|A\| = A \cdot A$ es la norma de una bolsa de palabras.

Por defecto, este modelo trabaja sobre palabras individuales, pero dispone de un parámetro `ngrams` que permite usar n-gramas especificando su tamaño. A la hora de entrenar los clasificadores, se ha ajustado mediante *kfold* para encontrar el mejor valor entre 1, 2, 3 y 4.

6.2.2. TF-IDF

TF-IDF es una técnica para ponderar la relevancia de términos en colecciones de textos que parte de los conceptos de frecuencia de un término (Luhn 1957) y frecuencia inversa de documento (Spärck Jones 1972). El objetivo de TF-IDF es restar valor a los términos que aparecen habitualmente en el lenguaje y remarcar los que puedan contener información semántica.

La frecuencia de un término en un documento (TF) es el número de apariciones del mismo dividido por el número de elementos de ese texto; es decir, cuantas más veces aparece un término en un texto, más alto es su TF. La frecuencia inversa de documento para un término es el número de textos en el conjunto dividido por el número de los que contienen el término; o sea, cuantos más textos contengan el término, menor será su IDF. TF-IDF se calcula para cada término en cada texto como el producto de TF e IDF. Por tanto, TF-IDF es mayor cuando un término aparece mucho en un texto pero poco en el conjunto.

El clasificador implementado toma todos los textos de entrenamiento y calcula los TF-IDF de cada uno. A continuación, se separan los de cada clase (promotor, no promotor) y se calcula la media de los valores de cada término. A la hora de clasificar un texto, para cada clase se suman los valores de TF-IDF medios de cada palabra presente. La clase con un total mayor es en la que se clasifica, pues en ella los términos tienen más relevancia.

Al igual que en el caso anterior, este modelo trabaja sobre palabras individuales por defecto, pero dispone de un parámetro `ngrams` que permite usar n-gramas especificando su tamaño. Al entrenar los clasificadores, se ha ajustado mediante *kfold* para encontrar el mejor valor entre 1, 2, 3 y 4.

6.2.3. FastText

FastText[1] es una herramienta desarrollada por Facebook (Joulin et al. 2016) para representar y clasificar texto mediante *word embeddings*. La clasificación se lleva a cabo mediante similitud coseno entre los vectores.

[1] https://fasttext.cc

FastText ofrece vectores preentrenados para diversos idiomas (Grave et al. 2018) y se ha usado el de español como base para el modelo. Este vectores tienen una dimensión de 300 componentes, pero se ha probado a reducirlos a 100 y a ajustar este parámetro mediante *kfold* para comprobar si afecta al resultado. Asímismo, se han ajustado los parámetros de ratio de aprendizaje entre 0.1, 0.5 y 1.0, el tamaño de los ngramas entre 1 y 2, y el número de iteraciones del entrenamiento entre 5, 25 y 50.

6.2.4. SpaCy

SpaCy[2] es una herramienta desarrollada por Explosion para implementar modelos de NLP de nivel industrial. De entre los tipos de modelo posibles se ha usado la opción *ensemble* que combina una bolsa de palabras con una red neuronal convolucional.

Como base del clasificador se han usado modelos preentrenados que ofrece SpaCy para español. Existen tres variantes de dicho modelo dependiendo de su tamaño, por lo que se ha usado el ajuste por *kfold* para seleccionar el que diese mejores resultados. Además, se ha ajustado el parámetro *iterations*, que regula el número de veces que el modelo itera sobre el corpus durante el entrenamiento, entre los valores 1, 5 y 10.

6.2.5. Transformers

HuggingFace Transformers[3] es una herramienta desarrollada por HuggingFace que permite implementar modelos basados en el concepto de `transformer` definido por Vaswani et al. (2017). HuggingFace ofrece un repositorio de modelos preentrenados para usar como base del aprendizaje, del que se ha usado `bert-base-multilingual-cased`[4], una adaptación del BERT de Google (Devlin et al. 2019).

Debido al coste temporal de entrenar este modelo, se ha optado por usar los parámetros por defecto sin ajustarlos.

[2] https://spacy.io
[3] https://huggingface.co/transformers
[4] https://huggingface.co/bert-base-multilingual-cased

7. Resultados

En este capítulo se tratan los experimentos realizados con los diferentes clasificadores y preprocesadores de texto, así como los resultados obtenidos y una discusión de los mejores clasificadores.

7.1. Métricas usadas

Por un lado, se han usado métricas de calidad de clasificación clásicas. Se dispone de un conjunto de escenarios y las clases a las que pertenecen. Para cada escenario se realiza una predicción de las clases a la que pertenece. Con las clasificaciones reales y las predichas se calcula una matriz de confusión para cada clase que contiene los positivos verdaderos, los positivos falsos, los negativos verdaderos y los negativos falsos, dependiendo de si dicha clase aparece o no en cada clasificación.

La *F1-score*, la *precision* (precisión) y el *recall* (sensibilidad) son métricas de la calidad con la que un clasificador predice una clase. La *F1-score* de una clase es la media armónica entre *precision* y *recall*. Representa la fiabilidad del clasificador en conjunto. La *precision* de una clase es la fracción de los positivos verdaderos entre la suma de positivos verdaderos y falsos. Representa el grado en que "son todos los que están". El *recall* de una clase es la fracción de los positivos verdaderos entre la suma de los positivos verdaderos y los negativos falsos. Representa el grado en que "están todos los que son".

Por otro lado, se ha medido la velocidad a la que los clasificadores procesan los textos en segundos partido por *tweet* y en *tweets* partido por segundo.

7.2. Entorno de ejecución de los experimentos

Los experimentos han sido realizados en el servidor HERMES del I3A, que ejecuta un Ubuntu 18.04.5 LTS con dos procesadores Intel Xeon E5-2630 de 8 núcleos cada uno y 64GB de RAM. A pesar de tener acceso a varias GPU Nvidia GeForce GTX 980 en el servidor no se ha conseguido hacerlas funcionar correctamente con las librerías usadas, por lo que todos los experimentos se han ejecutado sobre CPU. Futuras investigaciones deberán medir las mejoras obtenidas al realizar los experimentos sobre GPU. En el Anexo C se encuentran las versiones de todo el software usado.

Como se ha indicado anteriormente, los datos disponibles se dividen en proporción 80/20 para entrenamiento y validación. Por tanto se han usado 5436 textos para entrenamiento y 1360 para validación.

Es relevante destacar que los experimentos realizados no son completamente reproducibles. Esto se debe a que el entrenamiento de algunos de los modelos hace uso de funciones asíncronas multihilo por eficiencia. Esto resulta en que los eventos del sistema operativo generen entropía y los resultados no sean iguales. Ejecutar el entrenamiento en monohilo es inviable temporalmente, por lo que se ha continuado con la configuración por defecto. Sin embargo, se considera que la calidad de los resultados obtenidos es suficiente para extraer las conclusiones presentadas a continuación.

7.3. Coste temporal de los preprocesadores

Previamente se han detallado las distintas técnicas de preprocesado empleadas para reducir el ruido del texto antes de entrar al clasificador, incluyendo un prototipo de corrector ortográfico basado en distancias de edición. Cada una de las cinco cadenas de preprocesadores originales ha sido usada con y sin el corrector ortográfico, dando un total de diez cadenas usadas en los experimentos.

La complejidad del preprocesado de cada una de estas cadenas es diferente e impone costes temporales no triviales al clasificado, por lo que se ha estimado el tiempo en segundos que tarda cada una en tratar un *tweet*. Esta estimación se ha calculado midiendo el tiempo usado para preprocesar los 6796 textos del corpus Mix. Además, puesto que el corrector ortográfico dispone de una memoria caché interna para almacenar las correcciones conocidas, se ha realizado una segunda medición sobre el corpus para establecer la mejor velocidad con la caché llena.

En el Cuadro 7.1 se recogen los costes temporales medidos para cada cadena en la primera pasada de preprocesado (R1) y en la segunda (R2). Los identificadores reflejados se usan más adelante. A las cadenas de preprocesadores que han usado el corrector ortográfico se les añade la etiqueta `fix` y los valores de sus identificadores son pares.

Cuadro 7.1.: Coste temporal de las cadenas de preprocesadores

Id	**Preprocesadores**	**R1 (seg/tw)**	**R2 (seg/tw)**
C0	`no_newlines`, `no_url`, `fix`	0.15055	0.00116
C1	`no_newlines`, `no_url`	1e-05	1e-05
C2	`no_newlines`, `no_url`, `same_case`, `fix`	0.12953	0.00112
C3	`no_newlines`, `no_url`, `same_case`	2e-05	2e-05
C4	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`, `fix`	0.0774	0.0008
C5	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`	2e-05	2e-05
C6	`no_newlines`, `no_url`, `same_case`, `no_accents`, `only_letters`, `fix`	0.06138	0.00071
C7	`no_newlines`, `no_url`, `same_case`, `no_accents`, `only_letters`	6e-05	6e-05
C8	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`, `no_anorexia_bulimia`, `no_ana_mia`, `fix`	0.07864	0.00082
C9	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`, `no_anorexia_bulimia`, `no_ana_mia`	2e-05	2e-05

Se observa que las diferencias para una misma cadena de preprocesadores con y sin corrector ortográfico pueden llegar a los 4 órdenes de magnitud. Si bien se aprecia que el cacheado del corrector reduce notablemente el coste, lo cual sería de gran ayuda en un sistema en producción en el que se ven constantemente palabras tratadas previamente, en el mejor de los casos se sigue incrementando el tiempo en al menos un orden de magnitud.

También cabe destacar que el coste del corrector ortográfico desciende conforme se preprocesa más el texto de entrada. Esto se debe a que un mayor preprocesado previo implica una menor cantidad de términos válidos en el diccionario del corrector, y por tanto comparaciones más rápidas. Por el contrario, esto aumenta el coste temporal de las cadenas que no incluyen corrección.

Sin embargo, en el siguiente apartado se ve que el perjuicio del coste temporal que producen las cadenas de preprocesadores no puede ser evaluado por sí mismo, sino que depende del modelo de clasificador usado.

7.4. Métricas de los clasificadores

Tras entrenar y validar todas las combinaciones de modelos y cadenas de preprocesadores se han obtenido 50 clasificadores. Por simplicidad, se plasman en los cuadros únicamente el mejor resultado con y sin corrector ortográfico para cada modelo. En el Anexo D se encuentran los resultados completos de los experimentos.

En el Cuadro 7.2 se recogen los valores de *F1-score*, *precision* y *recall* ordenados de mejor a peor *F1-score* (la columna *Top* indica la posición en el ranking global).

Cuadro 7.2.: Métricas de los mejores resultados de cada modelo con y sin corrector ortográfico

Top	**Modelo**	**Cadena**	**F1-score**	**Precision**	**Recall**
1	Transformers	C4	0.960	0.963	0.957
2	Transformers	C3	0.958	0.956	0.960
6	FastText	C8	0.952	0.949	0.955
8	FastText	C5	0.948	0.946	0.951
16	SpaCy	C5	0.938	0.920	0.957
17	SpaCy	C4	0.936	0.952	0.921
31	Custom_BoW	C4	0.831	0.908	0.765
32	Custom_BoW	C5	0.822	0.902	0.755
34	Custom_TF-IDF	C6	0.818	0.782	0.858
35	Custom_TF-IDF	C5	0.818	0.786	0.852

En primer lugar, se observa que los modelos industriales (Transformers, FastText y SpaCy) tienen una clara ventaja sobre los modelos implementados manualmente (Custom_BoW y Custom_TF-IDF). en concreto, el *F1-score* del mejor Transformers es un 15,5 % mayor que el del mejor Custom_BoW. En segundo lugar, si bien los resultados de los modelos industriales están muy cerca unos de otros (el *F1-score* del mejor Transformers es un 5,4 % mayor que el del peor SpaCy), hay una clara ordenación, siendo Transformers el modelo que copa la cabeza de la tabla, seguido de FastText y dejando SpaCy a la cola. Además, se observa que el corrector ortográfico tiene una influencia positiva aunque muy reducida en la calidad de los resultados. Finalmente, se comprueba que la *precision* y el *recall* están muy equilibrados en los modelos industriales.

Por otro lado, en el Cuadro 7.3 se incluye para cada clasificador el coste temporal de su cadena de preprocesadores (valor de R2), el coste del modelo en sí medido sobre los datos de validación, y la cadencia de *tweets* por segundo que el clasificador en su conjunto es capaz de etiquetar.

Cuadro 7.3.: Coste temporal de los mejores clasificadores y cadencia de etiquetado

Top	Modelo	Cadena	T. cad. (seg/tw)	T. mod. (seg/tw)	Cadencia (tw/seg)
1	Transformers	C4	0.0008	0.10938	9
2	Transformers	C3	2e-05	0.11067	9
6	FastText	C8	0.00082	2e-05	1184
8	FastText	C5	2e-05	4e-05	16684
16	SpaCy	C5	2e-05	0.01144	87
17	SpaCy	C4	0.0008	0.01081	86
31	Custom_BoW	C4	0.0008	0.01564	60
32	Custom_BoW	C5	2e-05	0.01658	60
34	Custom_TF-IDF	C6	0.00071	2e-05	1364
35	Custom_TF-IDF	C5	2e-05	2e-05	25428

Se observa que existen grandes diferencias en el tiempo que usa cada modelo para clasificar el texto. Debido a ello, los modelos más rápidos como FastText y Custom_TF-IDF se ven muy lastrados a la hora de aplicar cadenas de preprocesadores con corrector ortográfico (un orden de magnitud menos de cadencia), mientras que esto apenas afecta a los otros modelos mucho más lentos. Como consecuencia, aunque las mejoras de *F1-score* al aplicar el corrector son similares para todos los modelos, el precio de la mejora es muy distinto.

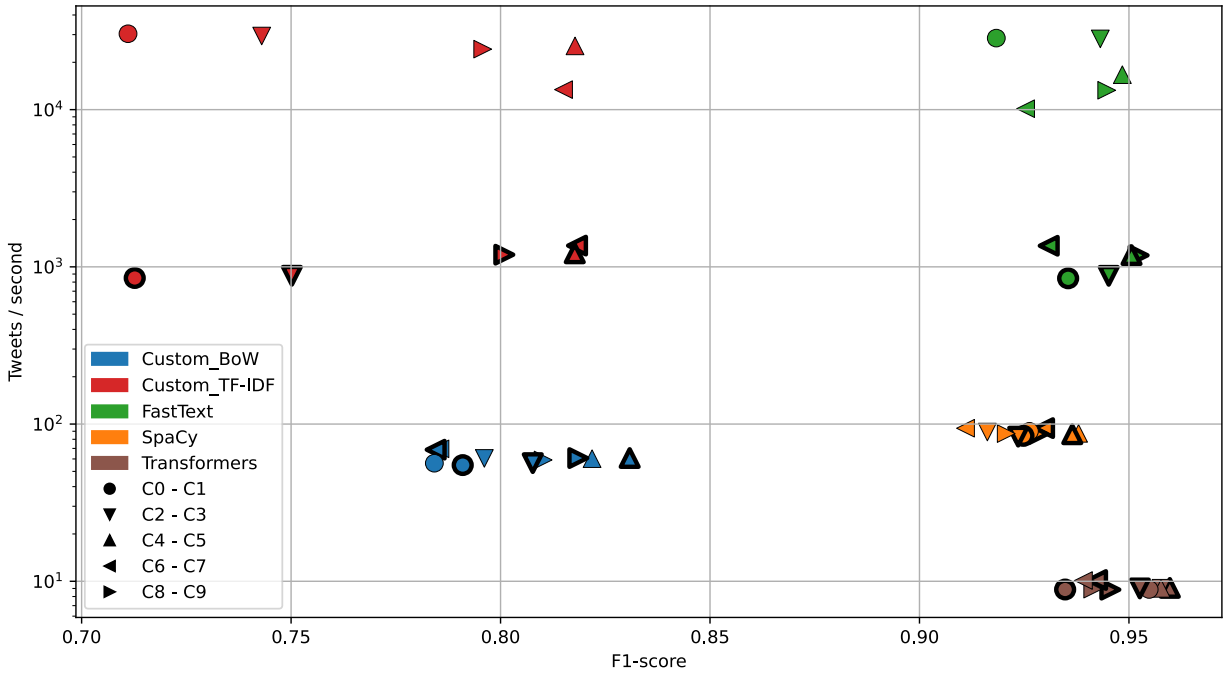

Figura 7.1.: Gráfica de dispersión de los valores de *F1-score* (X) y cadencia de clasificado (Y) de los clasificadores entrenados

En la Figura 7.1 se visualizan los valores de *F1-score* y cadencia (en escala logarítmica) para cada uno de los 50 clasificadores entrenados, siendo los iconos con borde grueso los clasificadores con corrector

ortográfico. En dicha gráfica se observan claramente todos los detalles indicados previamente, además de percibirse que los resultados de los clasificadores están completamente agrupados por modelos.

En cuanto al efecto de las cadenas de preprocesadores, se ha observado que cada modelo tiene un comportamiento distinto frente a ellas, tanto en si producen mejora como en su cuantía. La expresión más clara de esto es el gran incremento de *F1-score* que experimenta Custom_TF-IDF al ir aplicando preprocesadores frente a la mínima variación de Transformers. Un ejemplo más es que mientras que la cadena C1, que únicamente elimina URLs y saltos de línea, es de las mejores para Transformers, está a la cola en el resto de modelos. Otro ejemplo ilustrativo es que a pesar de que la cadena C9 ayuda a evitar sobreajuste de términos comunes (*ana*, *mia*, *anorexia* y *bulimia*) a FastText, Custom_BoW y Custom_TF-IDF, resulta contraproducente en Transformers y SpaCy. El único efecto común es el de las cadenas C4 y C5, que separan las palabras de los signos de puntuación, las cuales se posicionan a la cabeza de todos los modelos.

7.5. Explicación de modelos

Con el objetivo de comprender los razonamientos de cada modelo a la hora de clasificar textos se ha usado la herramienta LIME para generar explicaciones locales de varios textos del conjunto de validación. Dichas explicaciones asignan pesos a las distintas palabras del texto según su presencia aporte a que la clasificación sea promotora (naranja) o no promotora (azul). Como se ha indicado en capítulos anteriores, esta ponderación se obtiene realizando modificaciones en el texto y generando una regresión lineal de las clasificaciones resultantes.

Las explicaciones generadas han aportado una información muy limitada, seguramente debido a la corta extensión de los textos usados, pero permiten percibir algunos detalles de diferencia. A continuación se analizan las explicaciones generadas por cada uno de los cinco modelos con la cadena de preprocesadores C5 para el siguiente texto, extraído del conjunto usado para validación:

> En lugar de fumar un cigarrillo, prepara té verde. Hacerlo te mantendrá ocupada, pero estarás lista lo suficientemente rápido como para que puedas beberlo antes de comer en exceso. Tiene 0 calorías, cafeína y muchos beneficios pro-ana. Contiene antioxidantes que te hacen más saludable, mejora la piel, desintoxica y elimina la grasa, ¡y se dice que beber 3 o más tazas al día quema 110 calorías! Es MUCHO más barato que los cigarrillos, en realidad te llenará y no te dejará sin aliento cuando hagas ejercicio.

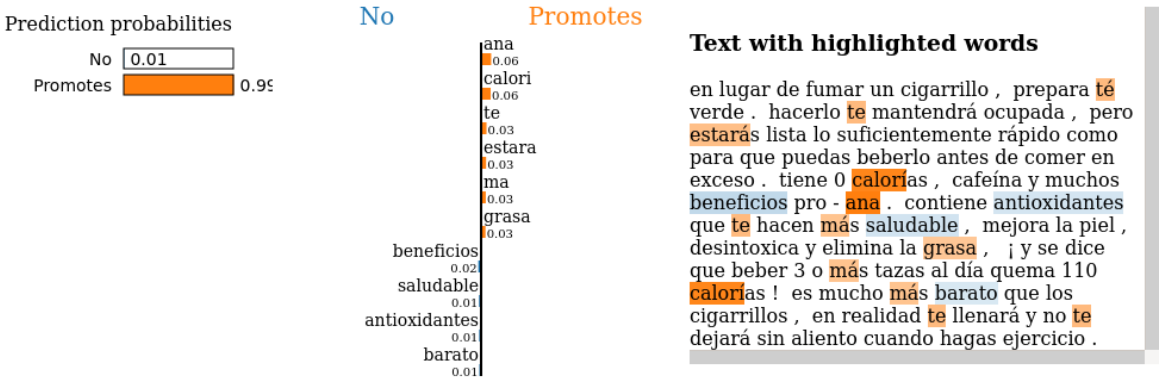

Figura 7.2.: Visualización LIME para Transformers con cadena C5

Por un lado, en las Figuras 7.2, 7.3 se observa que Transformers y FastText detectan claramente que el texto promueve la anorexia, y ambos clasificadores se fijan en palabras parecidas, como *ana*, *calorías* y *té*. Además, la diferencia de que FastText se fije en *comer* y Transformers no lo haga puede tener que

ver con la capacidad del Transformer de ignorar términos con semántica general, mientras que FastText se ha sobreajustado a los textos de entrenamiento donde aparece repetidamente dicho término.

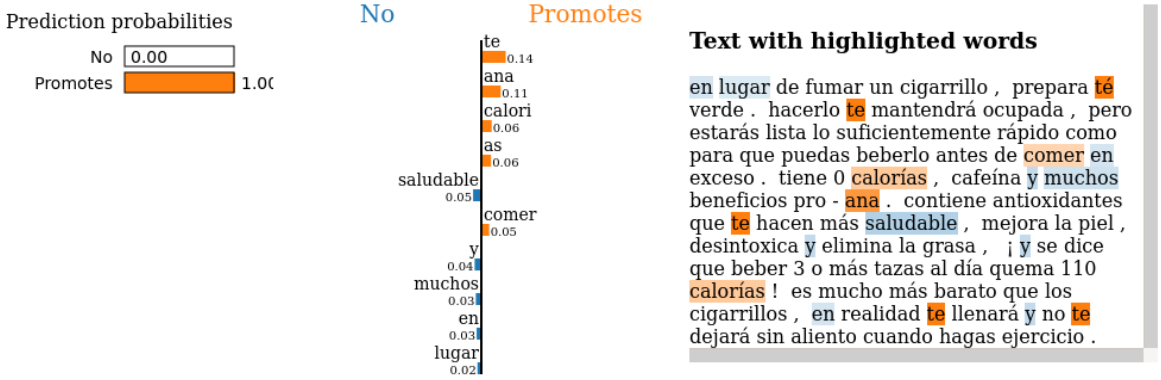

Figura 7.3.: Visualización LIME para FastText con cadena C5

Por otro lado, en las Figuras 7.4, 7.5 se identifica que Custom_BoW y Custom_TF-IDF tienen claros problemas de sobreajuste, ya que se fijan demasiado en preposiciones, lo que aumenta el porcentaje de indecisión a la hora de clasificar el texto. A pesar de ello son capaces de detectar algunos términos relevantes que les ayudan a decantarse por clasificar correctamente el texto.

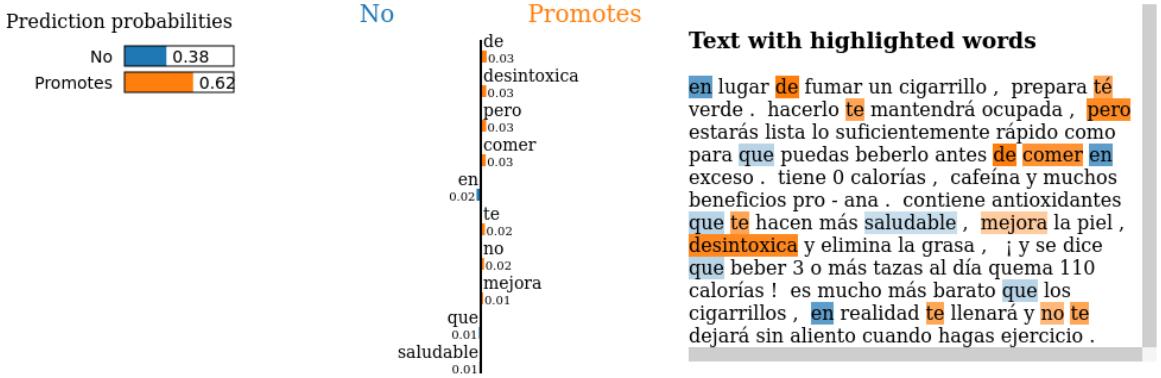

Figura 7.4.: Visualización LIME para Custom_BoW con cadena C5

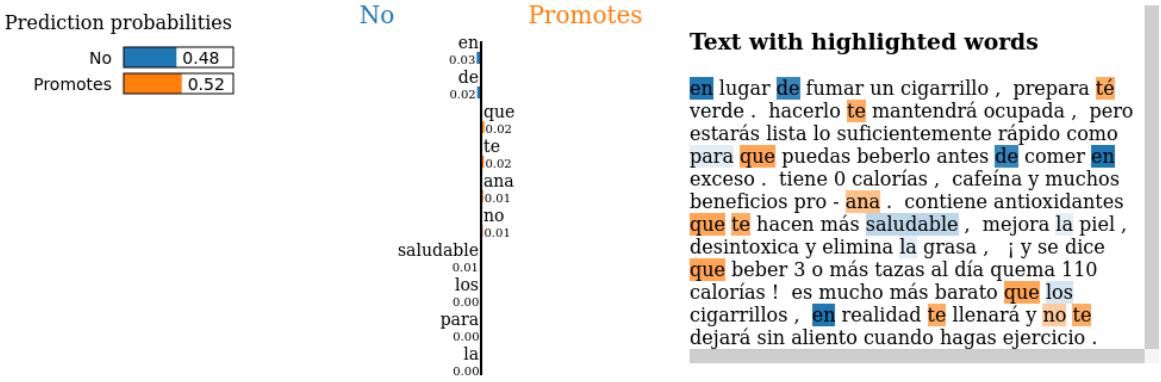

Figura 7.5.: Visualización LIME para Custom_TF-IDF con cadena C5

Por último, en la Figura 7.6 se ve que SpaCy está completamente equivocado en este caso y por algún motivo asigna erróneamente importancia a términos muy claros como *saludable* (marcado como promotor de la anorexia) y *ana* (marcado como no promotor).

En conjunto, se puede apreciar que los mejores resultados provienen de modelos capaces de ignorar el ruido del texto y centrarse en los términos con más contenido semántico. Tradicionalmente se ha

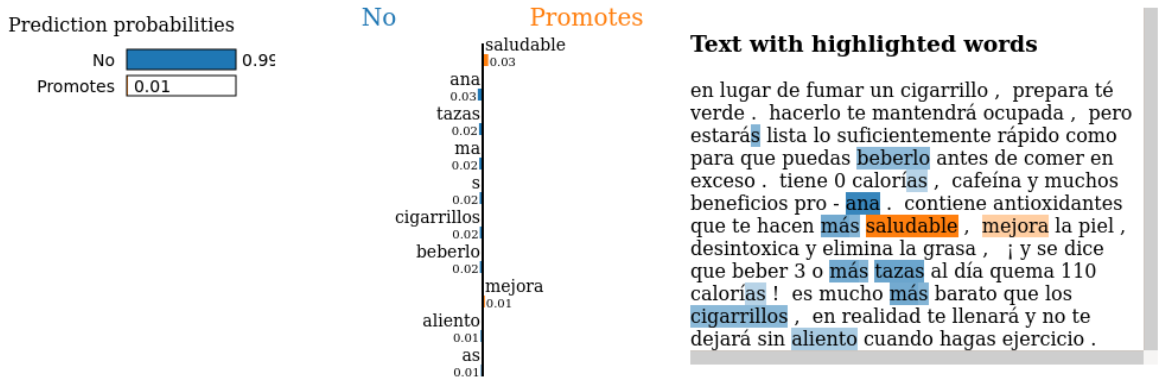

Figura 7.6.: Visualización LIME para SpaCy con cadena C5

optado por eliminar los términos comunes de cada idioma, conocidos como *stopwords*, para obtener mejorías en este aspecto, pero la introducción de mecanismos de atención como los Transformers ha permitido prescindir de ese tipo de cribado agresivo.

7.6. Discusión

En primer lugar, los datos obtenidos en los experimentos dejan claro que el refinamiento y el conocimiento volcados en el desarrollo de herramientas de uso industrial de código abierto dan lugar a una calidad de la clasificación notablemente mayor a lo que puede crear un profesional no experto en la materia sin muchos recursos dedicados. No sólo eso, sino que también superan los resultados de clasificadores implementados por profesionales expertos: los mejores valores de *F1-score* logrados en el trabajo de López-Úbeda et al. (2019) (con el que se comparte corpus de datos) son de tan sólo 0.916, lo que está por debajo de la práctica totalidad de los clasificadores industriales entrenados en este experiemnto.

Por otro lado, la decisión de qué clasificador es mejor va a depender del contexto de uso, en el cual se tendrá que dar peso a tres variables: *F1-score*, cadencia de clasificado y coste económico. El motivo de incluir el coste económico es que una inversión en GPUs va a mejorar en gran medida la cadencia de clasificado de los clasificadores basados en Transformers y sin embargo, es difícil predecir si estos resultados llegarían a superar a los de FastText con una inversión similar en mejorar la CPU. También en la línea de realizar una inversión económica para mejores prestaciones, y vista la calidad de un Transformer sencillo, podría plantearse hacer uso de la API que oferta OpenAI para clasificar textos mediante GPT-3[1], el modelo más avanzado de Transfomer existente a la fecha, y queda fuera del contexto del presente trabajo.

En definitiva, la disyuntiva práctica va a estar en si se necesita procesar una gran cantidad de documentos o si se requiere toda la precisión posible para unos pocos. En el ámbito concreto de este trabajo se requiere analizar el máximo posible de *tweets* en tiempo real para mapear interacciones en redes sociales, por lo que la respuesta es clara: el mejor clasificador para este proyecto es FastText con la cadena de preprocesadores C5 (no contiene corrector ortográfico).

Sin embargo, es importante recalcar que, como se detectó en la recogida de datos, existe una tendencia a los falsos positivos en el corpus usado, lo cual implica que los valores altos de *F1-score* no lo son tanto en realidad. A pesar de esta situación, la comparativa sigue siendo válida por los motivos expresados, ya que el sesgo se ha aplicado a todos los clasificadores por igual, incluidos los de López-Úbeda et al.

[1] Desde mediados de 2020 OpenAI oferta un servicio para hacer uso de GPT-3 a través de una API REST. En el momento de escribir este trabajo la API se encuentra en beta (véase https://beta.openai.com) y sólo se puede acceder previa petición de uso formal. La API se oferta mediante subscripción mensual cuyo coste oscila entre los 100 y 400 dólares.

Por otro lado, cabe plantearse si los resultados son extrapolables a otros contextos de clasificación binaria, o si por el contrario las herramientas estudiadas se comportan distinto dependiendo de la semántica a detectar. Por ejemplo, sería interesante saber si el mejor clasificador de textos promotores de la anorexia y la bulimia es también el idóneo para detectar textos que inciten al suicidio, mensajes de acoso o apología del terrorismo. Este tema puede ser otra línea de estudio futuro.

8. Conclusiones

En este trabajo se ha abordado la problemática de seleccionar e implementar un clasificador de texto capaz de detectar apología de trastornos alimenticios en redes sociales que se ha de integrar en el software que está desarrollando la Fundación APE para el seguimiento de este tipo de mensajes.

Se ha generado un corpus de textos etiquetados como promotores o no promotores de trastornos alimenticios expandiendo un corpus preexistente con mensajes recolectados de Internet. La selección y etiquetado se ha llevado a cabo mediante heurísticas (etiqueta común para todos los *tweets* con un mismo *hashtag* o para todas las publicaciones de un blog determinado), lo cual ha dado lugar a un porcentaje relevante de falsos positivos; estos errores de etiquetado afectan negativamente a la calidad de los clasificadores entrenados y a la capacidad de medir dicha calidad. El corpus generado ha sido usado para entrenar clasificadores de texto. Estos clasificadores están basados en cinco herramientas NLP distintas, dos de ellas implementadas manualmente como base de la comparativa. Además, se han aplicado distintas formas de preprocesado de texto, incluido un corrector ortográfico propio, para reducir el ruido en las muestras.

Los resultados obtenidos muestran una clara superioridad de las herramientas Transformers y FastText, basadas en *transformers* y *word embeddings* respectivamente, que han superado el 0.95 de *F1-score*. A pesar de que los resultados más altos han sido los de Transformers, FastText obtiene calidades comparables con una velocidad de procesado de varios órdenes de magnitud superior. Estos resultados son mucho mejores que los logrados por los clasificadores implementados manualmente en este trabajo y en otras publicaciones similares. En cuanto al preprocesado del texto, los resultados más consistentes se han logrado con técnicas poco intrusivas: paso a minúsculas y espaciado alrededor de signos de puntuación. Por su parte, la corrección ortográfica de los textos genera mejoras marginales en la calidad de la clasificación, mientras que añade un coste temporal relevante, por lo que se desaconseja su uso. Por tanto, la recomendación final es que la Fundación APE use FastText con preprocesado poco intrusivo.

En conjunto, se ha comprobado que con las herramientas disponibles actualmente es viable categorizar texto natural con un corpus reducido de ejemplos, sin hardware dedicado ni conocimiento extenso, lográndose unos resultados buenos. Esto abre las puertas a muchas entidades pequeñas y medianas para extraer información de redes sociales de una forma antes sólo posible para grandes organizaciones.

Trabajos futuros deben abordar cómo mejorar la calidad de los corpus de datos generados, ya sea mediante etiquetado manual o técnicas de detección de etiquetas erróneas, así como investigar si los resultados obtenidos son similares en otros ámbitos de clasificación de texto. También va a ser relevante estudiar si vale la pena hacer uso de las ofertas de *Machine Learning as a Service* que están surgiendo en la actualidad y van a continuar proliferando en los próximos años, como es el caso de la API de OpenAI.

Bibliografía

Amini, Hessam, y Leila Kosseim. 2020. «Towards Explainability in Using Deep Learning for the Detection of Anorexia in Social Media». En *Natural Language Processing and Information Systems*, editado por Elisabeth Métais, Farid Meziane, Helmut Horacek, y Philipp Cimiano, 225-35. Lecture Notes en Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-51310-8_21.

Aragon, Mario, Adrián López-Monroy, y Manuel Montes. 2019. «INAOE-CIMAT at eRisk 2019: Detecting Signs of Anorexia Using Fine-Grained Emotions». En *CLEF 2019*. Lugano, Switzerland.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. «Language Models Are Few-Shot Learners». *arXiv:2005.14165 [cs]*, julio. http://arxiv.org/abs/2005.14165.

Chorowski, Jan, Dzmitry Bahdanau, Kyunghyun Cho, y Yoshua Bengio. 2014. «End-to-End Continuous Speech Recognition Using Attention-Based Recurrent NN: First Results». En *NIPS 2014 Deep Learning Workshop*.

Chorowski, Jan, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, y Yoshua Bengio. 2015. «Attention-Based Models for Speech Recognition». En *Advances in Neural Information Processing Systems*.

Damerau, Fred J. 1964. «A Technique for Computer Detection and Correction of Spelling Errors». *Communications of the ACM* 7 (3): 171-76. https://doi.org/10.1145/363958.363994.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, y Kristina Toutanova. 2019. «BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding». *arXiv:1810.04805 [cs]*. http://arxiv.org/abs/1810.04805.

Dreisbach, Caitlin, Theresa A. Koleck, Philip E. Bourne, y Suzanne Bakken. 2019. «A Systematic Review of Natural Language Processing and Text Mining of Symptoms from Electronic Patient-Authored Text Data». *International Journal of Medical Informatics* 125: 37-46. https://doi.org/10.1016/j.ijmedinf.2019.02.008.

Goldberg, Yoav, y Omer Levy. 2014. «Word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method». *arXiv:1402.3722 [cs, stat]*, febrero. http://arxiv.org/abs/1402.3722.

Grave, Edouard, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, y Tomas Mikolov. 2018. «Learning Word Vectors for 157 Languages». *arXiv:1802.06893 [cs]*, marzo. http://arxiv.org/abs/1802.06893.

Hochreiter, Sepp, y Jürgen Schmidhuber. 1997. «Long Short-Term Memory». *Neural Computation* 9 (8): 1735-80. https://doi.org/10.1162/neco.1997.9.8.1735.

Hopfield, John. 1982. «Neural Networks and Physical Systems with Emergent Collective Computational Abilities». *Proceedings of the National Academy of Sciences of the United States of America* 79: 2554-58. https://doi.org/10.1073/pnas.79.8.2554.

Joulin, Armand, Edouard Grave, Piotr Bojanowski, y Tomas Mikolov. 2016. «Bag of Tricks for Efficient Text Classification». *arXiv:1607.01759 [cs]*, agosto. http://arxiv.org/abs/1607.01759.

Jurafsky, Daniel, y James Martin. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Second. Prentice-Hall.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, y L. D. Jackel. 1989. «Backpropagation Applied to Handwritten Zip Code Recognition». *Neural Computation* 1 (4): 541-51. https://doi.org/10.1162/neco.1989.1.4.541.

Lladó, Gina, Rocio Gonzalez-Soltero, y María Valderrama. 2017. «Anorexia y Bulimia Nerviosas: Difusión Virtual de La Enfermedad Como Estilo de Vida». *Nutrición Hospitalaria* 34 (junio): 693. https://doi.org/10.20960/nh.469.

López-Úbeda, Pilar, Miriam Plaza-del-Arco, Manuel Carlos Díaz-Galiano, L. Alfonso Ureña-López, y Maria-Teresa Martín-Valdivia. 2019. «Detecting Anorexia in Spanish Tweets». En *Proceedings - Natural Language Processing in a Deep Learning World*, 655-63. Incoma Ltd., Shoumen, Bulgaria. https://doi.org/10.26615/978-954-452-056-4_077.

Losada, David E., Fabio Crestani, y Javier Parapar. 2018. «Overview of eRisk: Early Risk Prediction on the Internet». En *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, editado por Patrice Bellot, Chiraz Trabelsi, Josiane Mothe, Fionn Murtagh, Jian Yun Nie, Laure Soulier, Eric SanJuan, Linda Cappellato, y Nicola Ferro, 343-61. Lecture Notes en Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-98932-7_30.

Luhn, H. P. 1957. «A Statistical Approach to Mechanized Encoding and Searching of Literary Information». *IBM Journal of Research and Development* 1 (4): 309-17. https://doi.org/10.1147/rd.14.0309.

Lundberg, Scott M., y Su-In Lee. 2017. «A Unified Approach to Interpreting Model Predictions». En *Advances in Neural Information Processing Systems 30*, editado por I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, y R. Garnett, 4765-74. Curran Associates, Inc.

Mikolov, Tomas, Kai Chen, Greg Corrado, y Jeffrey Dean. 2013. «Efficient Estimation of Word Representations in Vector Space». *arXiv:1301.3781 [cs]*, septiembre. http://arxiv.org/abs/1301.3781.

Mohammadi, Elham, Hessam Amini, y Leila Kosseim. 2019. «Quick and (Maybe Not so) Easy Detection of Anorexia in Social Media Posts». En *CLEF 2019*. Lugano, Switzerland.

Northcutt, Curtis G., Lu Jiang, y Isaac L. Chuang. 2020. «Confident Learning: Estimating Uncertainty in Dataset Labels». *arXiv:1911.00068 [cs, stat]*, febrero. http://arxiv.org/abs/1911.00068.

Ortega-Mendoza, Rosa María, D. I. H. Farías, y M. Montes-y-Gómez. 2019. «LTL-INAOE's Participation at eRisk 2019: Detecting Anorexia in Social Media through Shared Personal Information». En *CLEF 2019*. Lugano, Switzerland.

Radford, A., Jeffrey Wu, R. Child, David Luan, Dario Amodei, y Ilya Sutskever. 2019. «Language Models Are Unsupervised Multitask Learners». https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

Ramírez-Cifuentes, Diana, Christine Largeron, Julien Tissier, Ana Freire, y Ricardo Baeza-Yates. 2020. «Enhanced Word Embeddings for Anorexia Nervosa Detection on Social Media». En *Advances in Intelligent Data Analysis XVIII*, editado por Michael R. Berthold, Ad Feelders, y Georg Krempl, 404-17. Lecture Notes en Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-44584-3_32.

Ranganathan, Akshaya. 2019. «Early Detection of Anorexia Using RNN-LSTM and SVM Classifiers». En *CLEF 2019*, 11. Lugano, Switzerland.

Revillo Rey, Daniel. 2020. «Infraestructura software para el análisis de las interacciones en Twitter: aplicación a la detección de trastornos de conducta alimentaria». {Trabajo de Fin de Grado en Ingenier\'ia Inform\'atica}, Zaragoza: Universidad de Zaragoza.

Ribeiro, Marco Tulio, Sameer Singh, y Carlos Guestrin. 2016. «"Why Should I Trust You?": Explaining the Predictions of Any Classifier». *arXiv:1602.04938 [cs, stat]*, agosto. http://arxiv.org/abs/1602.04938.

Rumelhart, David E., Geoffrey E. Hinton, y Ronald J. Williams. 1986. «Learning Representations by Back-Propagating Errors». *Nature* 323 (6088): 533-36. https://doi.org/10.1038/323533a0.

Spärck Jones, Karen. 1972. «A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL». *Journal of Documentation* 28 (1): 11-21. https://doi.org/10.1108/eb026526.

Spinczyk, Dominik, Mateusz Bas, Mariusz Dzieciątko, Michał Maćkowski, Katarzyna Rojewska, y Stella Maćkowska. 2020. «Computer-Aided Therapeutic Diagnosis for Anorexia». *BioMedical Engineering OnLine* 19 (1): 53. https://doi.org/10.1186/s12938-020-00798-9.

Statista. 2020. «Total Data Volume Worldwide 2010-2024». https://www.statista.com/statistics/871513/worldwide-data-created/.

Trotzek, Marcel, Sven Koitka, y Christoph M Friedrich. 2018. «Word Embeddings and Linguistic Metadata at the CLEF 2018 Tasks for Early Detection of Depression and Anorexia». En *CLEF 2018*, 15. Avignon, France.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, y Illia Polosukhin. 2017. «Attention Is All You Need». *arXiv:1706.03762 [cs]*, diciembre. http://arxiv.org/abs/1706.03762.

Yan, Hao, Ellen E. Fitzsimmons-Craft, Micah Goodman, Melissa Krauss, Sanmay Das, y Patricia Cavazos-Rehg. 2019. «Automatic Detection of Eating Disorder-Related Social Media Posts That Could Benefit from a Mental Health Intervention». *International Journal of Eating Disorders* 52 (10): 1150-56. https://doi.org/10.1002/eat.23148.

A. Repositorio de código

URI: https://gitlab.com/miguescri/ed-classifier

Lenguaje de programación: Python

Licencia: MIT

Estructura del repositorio:

```
ed-classifier
+-- dataset
|   +-- blogs
|   |   +-- n_aclafeba.rss
|   |   +-- n_a-c-l.rss
|   |   +-- n_aluba.rss
|   |   +-- n_atopdegym.rss
|   |   +-- n_dimequecomes.rss
|   |   +-- n_feacab.rss
|   |   +-- n_midietacojea.rss
|   |   +-- n_realfooding.rss
|   |   +-- n_saludminimalista.rss
|   |   +-- n_tca-aragon.rss
|   |   +-- p_alexsoftskin.rss
|   |   +-- p_anaymia.rss
|   |   +-- p_anaymiathinsp.rss
|   |   +-- p_blogdeprincesas.rss
|   |   +-- p_elpasoalafelicidad.rss
|   |   +-- p_elrenacerdelaprincesa.rss
|   |   +-- p_mipd.rss
|   |   +-- p_otroinmar.rss
|   |   +-- p_princesakilljoy.rss
|   |   +-- p_princesalorelei.rss
|   |   +-- p_princesaproana.rss
|   |   +-- p_princesitadecristal.rss
|   |   +-- p_proanaxblog.rss
|   |   +-- README.md
|   |   `-- sites.md
|   +-- blogs.csv
|   +-- blogs_manual.csv
|   +-- blogs_tweetfied.csv
|   +-- blogs_tweetfied_manual.csv
|   +-- create_manual_subsets.py
|   +-- metrics.py
|   +-- parser.py
|   +-- SAD
|   |   +-- README.md
```

```
|   |   ·-- Spanish_anorexia_dataset_SAD.csv
|   +-- sad.csv
|   +-- sad_manual.csv
|   ·-- tweetfier.py
+-- ed_classifier
|   +-- classify.py
|   +-- db.py
|   +-- evaluators
|   |   +-- __init__.py
|   |   +-- lime.py
|   +-- ingestors
|   |   +-- __init__.py
|   |   +-- README.md
|   |   ·-- trawling.py
|   +-- __init__.py
|   +-- models
|   |   +-- custom_bow.py
|   |   +-- custom_tfidf.py
|   |   +-- fasttext.py
|   |   +-- __init__.py
|   |   +-- README.md
|   |   +-- spacy.py
|   |   ·-- transformers.py
|   +-- preprocessors
|   |   +-- __init__.py
|   |   ·-- README.md
|   +-- train.py
|   +-- utils.py
|   ·-- visualize.py
+-- LICENSE
+-- README.md
+-- requirements.txt
·-- trained
    ·-- download.py
```

B. Webs origen del corpus manual

Webs promotoras de desórdenes alimenticios:

- https://proanaxblog.wordpress.com/
- https://ana-y-mia-para-princesas7.webnode.es/
- https://otroinmarcesibleanayrexmas.wordpress.com/
- https://blogdeprincesasanamiayalisaprincipesrexybill.wordpress.com/
- https://elrenacerdelaprincesalorelei.wordpress.com/
- https://princesaloreleiproanapromia.wordpress.com/
- https://elpasoalafelicidad.wordpress.com/
- https://princesakilljoy.wordpress.com/
- https://elprincesitadecristal.wordpress.com/
- https://princesaproanaypromia.wordpress.com/
- https://anaymia1.home.blog/
- https://alexsoftskin.wordpress.com/
- https://carrerasdeayunoana.blogspot.com/
- https://carreraana.blogspot.com/
- http://princesas-de-ana-y-mia.over-blog.com
- https://mipd.blogspot.com/
- http://anaymiathinsp.blogspot.com/

Webs no promotoras:

- https://www.midietacojea.com/
- https://www.dimequecomes.com/
- https://www.saludminimalista.com/
- https://atopedegym.com/
- https://realfooding.com/
- https://feacab.org/
- https://aluba.org.ar/
- https://www.aclafeba.org/
- https://a-c-l-anorexia.blogspot.com/
- https://www.tca-aragon.org/

C. Versiones de software usado

Versión de Python:

```
Python 3.8.0
```

Librerías de Python:

```
abydos==0.5.0
beautifulsoup4==4.9.3
blis==0.7.4
cachetools==4.2.0
catalogue==1.0.0
certifi==2020.12.5
chardet==3.0.4
click==7.1.2
cycler==0.10.0
cymem==2.0.5
cytoolz==0.11.0
decorator==4.4.2
deprecation==2.1.0
es-core-news-lg @ https://github.com/explosion/spacy-models/releases/download/
        es_core_news_lg-2.3.1/es_core_news_lg-2.3.1.tar.gz
es-core-news-md @ https://github.com/explosion/spacy-models/releases/download/
        es_core_news_md-2.3.1/es_core_news_md-2.3.1.tar.gz
es-core-news-sm @ https://github.com/explosion/spacy-models/releases/download/
        es_core_news_sm-2.3.1/es_core_news_sm-2.3.1.tar.gz
fasttext==0.9.2
feedparser==6.0.2
filelock==3.0.12
idna==2.10
imageio==2.9.0
jellyfish==0.8.2
joblib==0.17.0
kiwisolver==1.3.1
lime==0.2.0.1
lxml==4.6.2
matplotlib==3.3.3
murmurhash==1.0.5
networkx==2.5
numpy==1.19.4
packaging==20.7
Pillow==8.0.1
pkg-resources==0.0.0
plac==1.1.3
```

```
preshed==3.0.5
pybind11==2.6.1
pyemd==0.5.1
pyparsing==2.4.7
Pyphen==0.10.0
python-dateutil==2.8.1
python-Levenshtein==0.12.0
PyWavelets==1.1.1
pyxDamerauLevenshtein==1.6.1
regex==2020.11.13
requests==2.25.0
sacremoses==0.0.43
scikit-image==0.17.2
scikit-learn==0.23.2
scipy==1.5.4
sgmllib3k==1.0.0
six==1.15.0
soupsieve==2.1
spacy==2.3.4
SQLAlchemy==1.3.20
srsly==1.0.5
textacy==0.10.1
textdistance==4.2.0
thinc==7.4.5
threadpoolctl==2.1.0
tifffile==2020.12.8
tokenizers==0.9.4
toolz==0.11.1
torch==1.7.1+cpu
tqdm==4.54.1
transformers==4.0.1
typing-extensions==3.7.4.3
Unidecode==1.1.1
urllib3==1.26.2
wasabi==0.8.0
```

D. Tablas resultados entrenamiento

Cuadro D.1.: Costes temporales de todos los preprocesadores

Id	Preprocesadores	R1 (seg/tw)	R2 (seg/tw)
0	`no_newlines`, `no_url`, `fix`	0.15055	0.00116
1	`no_newlines`, `no_url`	1e-05	1e-05
2	`no_newlines`, `no_url`, `same_case`, `fix`	0.12953	0.00112
3	`no_newlines`, `no_url`, `same_case`	2e-05	2e-05
4	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`, `fix`	0.0774	0.0008
5	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`	2e-05	2e-05
6	`no_newlines`, `no_url`, `same_case`, `no_accents`, `only_letters`, `fix`	0.06138	0.00071
7	`no_newlines`, `no_url`, `same_case`, `no_accents`, `only_letters`	6e-05	6e-05
8	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`, `no_anorexia_bulimia`, `no_ana_mia`, `fix`	0.07864	0.00082
9	`no_newlines`, `no_url`, `same_case`, `separate_punctuation`, `no_anorexia_bulimia`, `no_ana_mia`	2e-05	2e-05

Cuadro D.2.: Resultados de todos los entrenamientos

Top	Modelo	Cadena	F1-score	Precision	Recall
1	Transformers	4	0.95975	0.96273	0.95679
2	Transformers	3	0.95766	0.95545	0.95988
3	Transformers	5	0.95766	0.95545	0.95988
4	Transformers	1	0.95483	0.96384	0.94599
5	Transformers	2	0.9526	0.94394	0.96142
6	FastText	8	0.95231	0.94939	0.95525
7	FastText	4	0.95062	0.95062	0.95062
8	FastText	5	0.94842	0.94624	0.95062
9	Transformers	8	0.94557	0.95298	0.93827

Top	Modelo	Cadena	F1-score	Precision	Recall
10	FastText	2	0.94517	0.9459	0.94444
11	FastText	9	0.9447	0.94037	0.94907
12	FastText	3	0.94316	0.93884	0.94753
13	Transformers	6	0.94238	0.96446	0.9213
14	Transformers	9	0.94136	0.95404	0.92901
15	Transformers	7	0.93913	0.96272	0.91667
16	SpaCy	5	0.93797	0.91988	0.95679
17	SpaCy	4	0.93647	0.95215	0.9213
18	FastText	0	0.93548	0.93119	0.93981
19	Transformers	0	0.93477	0.92977	0.93981
20	FastText	6	0.93085	0.9374	0.92438
21	SpaCy	6	0.92973	0.93045	0.92901
22	SpaCy	8	0.92834	0.95888	0.89969
23	SpaCy	1	0.92625	0.95269	0.90123
24	FastText	7	0.92533	0.9232	0.92747
25	SpaCy	0	0.92482	0.97114	0.88272
26	SpaCy	2	0.92357	0.95395	0.89506
27	SpaCy	9	0.92071	0.91859	0.92284
28	FastText	1	0.91834	0.91692	0.91975
29	SpaCy	3	0.9162	0.94876	0.8858
30	SpaCy	7	0.91099	0.94825	0.87654
31	Custom_BoW	4	0.83082	0.90842	0.76543
32	Custom_BoW	5	0.82185	0.90221	0.75463
33	Custom_BoW	8	0.81857	0.90317	0.74846
34	Custom_TF-IDF	6	0.81825	0.782	0.85802
35	Custom_TF-IDF	5	0.81778	0.78632	0.85185
36	Custom_TF-IDF	4	0.81771	0.78359	0.85494
37	Custom_TF-IDF	7	0.81503	0.77997	0.8534
38	Custom_BoW	9	0.81017	0.8985	0.73765
39	Custom_BoW	2	0.80769	0.77557	0.84259
40	Custom_TF-IDF	8	0.8009	0.78038	0.82253
41	Custom_BoW	3	0.79613	0.76868	0.82562
42	Custom_TF-IDF	9	0.79575	0.78326	0.80864
43	Custom_BoW	0	0.79097	0.74897	0.83796
44	Custom_BoW	7	0.78553	0.86322	0.72068

Top	Modelo	Cadena	F1-score	Precision	Recall
45	Custom_BoW	6	0.78451	0.86296	0.71914
46	Custom_BoW	1	0.7842	0.74548	0.82716
47	Custom_TF-IDF	2	0.75019	0.72993	0.7716
48	Custom_TF-IDF	3	0.74299	0.73025	0.75617
49	Custom_TF-IDF	0	0.71268	0.69343	0.73302
50	Custom_TF-IDF	1	0.71108	0.71384	0.70833

Cuadro D.3.: Coste temporal de todos los clasificadores

Top	Modelo	Cadena	T. cadena (seg/tw)	T. modelo (seg/tw)	Cadencia (tw/seg)
1	Transformers	4	0.0008	0.10938	9
2	Transformers	3	2e-05	0.11067	9
3	Transformers	5	2e-05	0.11094	9
4	Transformers	1	1e-05	0.11273	8
5	Transformers	2	0.00112	0.10994	9
6	FastText	8	0.00082	2e-05	1184
7	FastText	4	0.0008	4e-05	1188
8	FastText	5	2e-05	4e-05	16684
9	Transformers	8	0.00082	0.11223	8
10	FastText	2	0.00112	2e-05	876
11	FastText	9	2e-05	5e-05	13278
12	FastText	3	2e-05	2e-05	28181
13	Transformers	6	0.00071	0.09746	10
14	Transformers	9	2e-05	0.11223	8
15	Transformers	7	6e-05	0.09882	10
16	SpaCy	5	2e-05	0.01144	87
17	SpaCy	4	0.0008	0.01081	86
18	FastText	0	0.00116	2e-05	845
19	Transformers	0	0.00116	0.11164	8
20	FastText	6	0.00071	2e-05	1361
21	SpaCy	6	0.00071	0.0099	94
22	SpaCy	8	0.00082	0.01077	86
23	SpaCy	1	1e-05	0.01111	89
24	FastText	7	6e-05	4e-05	10140
25	SpaCy	0	0.00116	0.01073	84
26	SpaCy	2	0.00112	0.0109	83

Top	Modelo	Cadena	T. cadena (seg/tw)	T. modelo (seg/tw)	Cadencia (tw/seg)
27	SpaCy	9	2e-05	0.01147	86
28	FastText	1	1e-05	2e-05	28505
29	SpaCy	3	2e-05	0.01122	88
30	SpaCy	7	6e-05	0.01057	94
31	Custom_BoW	4	0.0008	0.01564	60
32	Custom_BoW	5	2e-05	0.01658	60
33	Custom_BoW	8	0.00082	0.01563	60
34	Custom_TF-IDF	6	0.00071	2e-05	1364
35	Custom_TF-IDF	5	2e-05	2e-05	25428
36	Custom_TF-IDF	4	0.0008	2e-05	1218
37	Custom_TF-IDF	7	6e-05	2e-05	13407
38	Custom_BoW	9	2e-05	0.01689	59
39	Custom_BoW	2	0.00112	0.0166	56
40	Custom_TF-IDF	8	0.00082	2e-05	1195
41	Custom_BoW	3	2e-05	0.01652	60
42	Custom_TF-IDF	9	2e-05	2e-05	24254
43	Custom_BoW	0	0.00116	0.01707	54
44	Custom_BoW	7	6e-05	0.01432	69
45	Custom_BoW	6	0.00071	0.01384	68
46	Custom_BoW	1	1e-05	0.01773	56
47	Custom_TF-IDF	2	0.00112	2e-05	877
48	Custom_TF-IDF	3	2e-05	2e-05	29396
49	Custom_TF-IDF	0	0.00116	1e-05	849
50	Custom_TF-IDF	1	1e-05	2e-05	30390

Cuadro D.4.: Parámetros de todos los entrenamientos

Top	Modelo	Cadena	Parámetros
1	Transformers	4	`epoch`: 3
2	Transformers	3	`epoch`: 3
3	Transformers	5	`epoch`: 3
4	Transformers	1	`epoch`: 3
5	Transformers	2	`epoch`: 3
6	FastText	8	`epoch`: 50, `lr`: 0.5, `ngrams`: 2, `pretrained`: trained/cc.es.100.vec, `dim`: 100

Top	Modelo	Cadena	Parámetros
7	FastText	4	epoch: 25, lr: 1.0, ngrams: 2, pretrained: trained/cc.es.300.vec, dim: 300
8	FastText	5	epoch: 25, lr: 1.0, ngrams: 2, pretrained: trained/cc.es.300.vec, dim: 300
9	Transformers	8	epoch: 3
10	FastText	2	epoch: 5, lr: 0.5, ngrams: 1, pretrained: trained/cc.es.300.vec, dim: 300
11	FastText	9	epoch: 25, lr: 1.0, ngrams: 2, pretrained: trained/cc.es.300.vec, dim: 300
12	FastText	3	epoch: 50, lr: 0.5, ngrams: 1, pretrained: trained/cc.es.300.vec, dim: 300
13	Transformers	6	epoch: 3
14	Transformers	9	epoch: 3
15	Transformers	7	epoch: 3
16	SpaCy	5	iterations: 5, base_model: es_core_news_md
17	SpaCy	4	iterations: 10, base_model: es_core_news_md
18	FastText	0	epoch: 25, lr: 0.5, ngrams: 1, pretrained: trained/cc.es.300.vec, dim: 300
19	Transformers	0	epoch: 3
20	FastText	6	epoch: 5, lr: 0.5, ngrams: 2, pretrained: trained/cc.es.100.vec, dim: 100
21	SpaCy	6	iterations: 5, base_model: es_core_news_lg
22	SpaCy	8	iterations: 10, base_model: es_core_news_lg
23	SpaCy	1	iterations: 10, base_model: es_core_news_md
24	FastText	7	epoch: 5, lr: 1.0, ngrams: 2, pretrained: trained/cc.es.300.vec, dim: 300
25	SpaCy	0	iterations: 10, base_model: es_core_news_lg
26	SpaCy	2	iterations: 10, base_model: es_core_news_md
27	SpaCy	9	iterations: 10, base_model: es_core_news_lg
28	FastText	1	epoch: 50, lr: 1.0, ngrams: 1, pretrained: trained/cc.es.300.vec, dim: 300
29	SpaCy	3	iterations: 10, base_model: es_core_news_md
30	SpaCy	7	iterations: 5, base_model: es_core_news_md
31	Custom_BoW	4	ngram: 2
32	Custom_BoW	5	ngram: 2
33	Custom_BoW	8	ngram: 2
34	Custom_TF-IDF	6	ngram: 2
35	Custom_TF-IDF	5	ngram: 1

Top	Modelo	Cadena	Parámetros
36	Custom_TF-IDF	4	`ngram`: 1
37	Custom_TF-IDF	7	`ngram`: 2
38	Custom_BoW	9	`ngram`: 2
39	Custom_BoW	2	`ngram`: 2
40	Custom_TF-IDF	8	`ngram`: 1
41	Custom_BoW	3	`ngram`: 2
42	Custom_TF-IDF	9	`ngram`: 1
43	Custom_BoW	0	`ngram`: 2
44	Custom_BoW	7	`ngram`: 2
45	Custom_BoW	6	`ngram`: 2
46	Custom_BoW	1	`ngram`: 2
47	Custom_TF-IDF	2	`ngram`: 2
48	Custom_TF-IDF	3	`ngram`: 2
49	Custom_TF-IDF	0	`ngram`: 1
50	Custom_TF-IDF	1	`ngram`: 2